This monograph was first put together in March–June 2020
There was a corona virus pandemic at that time
This version was compiled July 6, 2020: 10:03 am

• This is the ‘coloured’ version of my monograph.
• It is the one without cross-references to the overhead/foil/slide/screen version.
Kari; Charlotte and Nikolaj; Camilla, Marianne, Katrine, Caroline and Jakob
Preface

The Triptych Dogma

In order to specify software, we must understand its requirements. In order to prescribe requirements, we must understand the domain. So we must study, analyse and describe domains.

General

The claim of this monograph is twofold:

• that domain engineering is a viable, yes, we would claim, necessary initial phase of software development; and
• that domain science & engineering is a worthwhile topic of research.

I mean this rather seriously:

• How can one think of implementing software, preferably satisfying some requirements, without demonstrating that one understands the domain?

So in this monograph I shall

• explain what domain engineering is,
• some of the science that goes with it, and
• how one can “derive” requirements prescriptions
  ∞ (for computing systems)
  ∞ from domain descriptions.

But there is an altogether different reason, also, for presenting these papers in monograph form:

• Software houses may not take up the challenge to develop software
  ∞ that satisfies customers expectations, that is, reflects the domain such as these customers know it,
  ∞ and software that is correct with respect to requirements, with proofs of correctness often having to refer to the domain.
• But computing scientists are shown, in these papers, that domain science and engineering is a field full of interesting problems to be researched.
• We consider domain descriptions, requirements prescriptions and software design specifications to be mathematical quantities.

Application Areas

Computers are man-made, they are artefacts. Physicists and engineers compute over domains of physics, including chemistry, and engineering designs, and their computations range mostly over phenomena of physics. Manufacturing, logistics and transport firms as well as goods importers/exporters, wholesalers and retail firms use computers significantly. Their domain is mostly operations research. With domain

1 cf. ‘Engineering’ in the main title and the second subtitle of this monograph
2 We use the ampersand ‘&’ to emphasize that domain science & engineering is one topic, not two.
3 cf. ‘Science’ in the main title and the first subtitle of this monograph
**science & engineering** the domain (of possible software applications) is now definable in terms of what the method of this monograph is capable of handling. Briefly, but by far not exhaustively, that domain includes such which focus on man-made objects, i.e., on artefacts, and the interaction of humans with these. In that respect the domain science & engineering, when used for the purposes of software development, straddles the aforementioned application areas but now, we claim, with some firm direction.

**Work in Progress**

The state of this monograph reflects that it is ‘*work in progress*’. The first publication directing the method in the direction of what is presented here were [53, 57, Summer 2010]. Since then there has been several publications in peer reviewed journals [70, 76, 74, 78, Years 2017–2019]. In the period of submission of the most recent of these [78, Spring 2018], and during the writing of this monograph, up to this very moment this Preface is being written, new research discoveries are made. The way that these new research ideas fits well within the framework, also in its detailed aspects, makes me think that the body of work presented here is stable and durable. I have therefore decided to release the monograph now in the hope that it might inspire others to continue the research.

**The Monograph as a Textbook**

Many universities appears to teach their science students, whether BSc or MSc, only such material for which there exists generally accepted and stable theories. I have over the years, since 1976, when I first joined a university staff — then as a full professor — mostly not adhered to this limitation, but taught, to BSc/MSc students, such material that yet had to reach the maturity of a *scientific theory*. So, go ahead, use this monograph in teaching!

**Specific**

This monograph is intended at the following mathematics–minded audiences:

- primarily researchers, lecturers and PhD students in the sciences of computers and computing — conventionally speaking: those who have few preconceived objections to the use of discrete mathematics;
- hopefully also their similarly oriented curious and serious MSc students;
- and finally, recent, and not so recent, practicing software engineers and programmers — again open-minded with respect to new foundations for programming and formalisms.

At the end of most chapters’ ‘Problem Exercise’ sections, we suggest a number of anywhere from engineering to science challenges: project-oriented domain analysis & description class-project exercises as well as more individual research problems of more-or-less “standard” degree of difficulty to plain challenging studies. The class-project exercises amount to rather “full-scale” 4–6 student term projects.

**Sources**

This is a monograph of 11 chapters. Except for three (Chapters 0, 2 and 10), these chapters build on the following publications:

- **Chapter 1: Philosophy** [77] 11–16
- **Chapter 3: External Qualities** [78, Sects. 2–3] 39–83
- **Chapter 4: Internal Qualities** [78, Sect. 4] 85–122
- **Chapter 5: Transcendental Deduction** [78, Sects. 5–6] and [77] 123–124

[70] is a precursor for [78]
Chapters 0–2 pave the way. They introduce the reader to a vocabulary of concepts specific to computing science; to some fundamental ideas of philosophy – a new to any treatise of our field; and to prerequisite concepts of discrete mathematics, of space, time and matter, and of unique identification and mereology – also new to any treatise of our field.

Chapters 3–6 form the real core of this monograph. It is here we develop what we shall, unashamedly, refer to as both a science and an engineering, i.e., a methodology for understanding the concept of ‘domains’ such as we shall define it. These chapters study and develop calculi for the analysis of domains and for their description. At the same time as presenting this study these chapters also present a method for actually developing domain descriptions. This duality, the beginnings of a scientific, theoretical foundation for domain analyser & describer, and the beginnings of a method for actual engineering development, may seem confusing if the twin aspects are not kept clear from one another. We have endeavoured to present the two aspects reasonable separated.

Chapters 7–9 are “bonus” chapters! They contain some quite original concepts: domain facets (Chapter 7) such as intrinsics, support technology, rules & regulations, scripts, license languages, management & organisation and human behaviour; requirements engineering (Chapter 8) concepts such as the distinction into domain requirements, interfaces requirements and machine requirements, projection, instantiation, determination, extension and fitting, and more – not quite the way conventional requirements engineering textbooks treat the field; and demos, simulators, monitors and controllers (Chapter 9) are all concepts that, we claim, can be interestingly understood in light of domain descriptions being developed into requirements prescriptions and these into software designs and software. These chapters may, for better or worse, not be of interest to some computer scientists, but should be of interest to software engineering practitioners and people who do study the more mundane aspects of software engineering.

Some Caveats

This monograph uses the RAISE Specification Language, RSL [179, 176] for its formal presentations and for its mixed mathematical notation and RSL informal explanations. We refer to Appendix C for a résumé of RSL. [177, 168, 172] provide short, concise introductions to the RAISE Method and to RSL. Equally relevant other specification languages could be VDM SL [88, 89, 154], Z [374], the B Method notation [1], Alloy [251], and others. Also algebraic approaches are possible, for example: CafeOBJ [283, 126], CASL [128] or Maude [283, 126]. Lecturer and students, readers in general, perhaps more familiar with some of the above languages than with RSL, should be able to follow our presentations, but perform their exercise/term project work in the language of their choice.

This monograph is the first in which domain science & engineering is presented in a coherent form, ready for scientific study as well as for university classes. But it is far from a polished textbook: Not all “corners” of describable, manifest and artefactual domains are here given “all the necessary” principles, techniques and [language] tools necessary for “run-of-the-mill” software development. We have given sufficiently many university courses, over previous texts, and these have shown, we claim, that most students can be expected, under guidance of professionals experienced in formal specifications, to contribute meaningfully to professional domain analysis & description projects.

We have left out of this monograph potential chapters on for example: possible Semantic Models of the domain analysis & description calculi [62]. We invite the reader to study this reference as well as to contribute to domain science. Examples of the latter could, for example, entail: A Study of Analysis & Description Calculi: on the order of analysis & description prompts; on the top-down analysis & description, as suitable, for artefactual domains versus bottom-down analyses & descriptions, as perhaps more suitable, for natural and living specific domains, including humans; a deeper understanding of Intentional Pull, et cetera.
Acknowledgments

This is most likely the last book that I may be able to publish. Over the years I have co-edited, edited, co-authored or authored a number of published books. Some more noteworthy are: [88, 32, 92, 89, 33, 83, 19, 90, 39, 40, 41, 79, 45, 46, 47, 49, 50, 51].

Over all these years I have benefited in my research from a large number of wonderful people. I bear tribute, in approximate chronological order, to a few of these: (the late) Cai Kinberg, Gunnar Wedell, (the late) Jean Paul Jacob, Peter Johansen, (the late) Gerald Weinberg, (the late) Lotfi Zadeh, (the late) Ted Codd, (the late) John W. Backus, (the late) Peter Lucas, Cliff Jones, (the late) Hans Bekić, Kurt Walk, (the late) Christian Gram, Hans Bruun, Andrzej Blikle, Dömöldi Balint, (the late) Asger Kjerbye Nielsen, Josef Gruska, Ivan Havel, Erich Neuhold, Ole N. Oest, (the late) Søren Prehn, Michael A. Jackson, Sir Tony Hoare, Hans Langmaack, Larry E. Druffel, Enn S. Tyugu, Dominique Méry, Zhou Chao Chen, Kokichi Futatsugi, Enn S. Tyugu, Chris George and Klaus Havelund.

I also wish to thank the many colleagues around the world who have let me try out the ideas of earlier versions of this monograph on their students in MSc/PhD courses since my retirement, at age 70, in 2007: Egon Börger (Pisa), Dominique Méry (Nancy), Wolfgang J. Paul (Saarbrücken), Alan Bundy (Edinburgh), Dömöldi Balint (Budapest), Andreas Hamfeldt (Uppsala), Luís Barbosa (Braga), Jin Song Dong (Singapore and Brisbane), Jens Knoop (Vienna), Magne Haveraaen (Bergen), Zhu Huibiao (Shanghai), Chin Wei Ngan (Singapore) and Zhang Min (Shanghai).

In thank my Springer editor Ronan Nugent for steadfast and caring encouragement.

I finally wish to thank Kai Sørlander for his Philosophy [345, 346, 347, 348]. As you shall find out, Sørlander’s Philosophy has inspired me tremendously. Ideas that were previously vague, are now, to me, crystal clear. I hope you will be likewise enlightened.

Dines Bjørner. July 6, 2020: 10:03 am
Fredsvej 11, DK–2840 Holte, Denmark
2 Logic and Mathematics

Identity and Mereology .................................. 17
2.1 Prologue ............................................. 17
2.2 Logic .................................................. 17
2.3 Mathematics .......................................... 17
  2.3.1 Mathematical Logic .............................. 18
  2.3.2 Sets ............................................. 19
  2.3.3 Types ............................................ 19
  2.3.4 Functions ........................................ 20
    2.3.4.1 Total and Partial Functions ............... 20
    2.3.4.2 Predicate Functions ....................... 20
  2.3.5 Mathematical Notation versus Formal
       Specification Languages ........................ 20
    2.3.5.1 Mathematics as a Notation – In General ... 20
    2.3.5.2 Mathematics as a Notation – In Specific .. 21
      2.3.5.2.1 The Dot-dot-dot Notation............. 21
      2.3.5.2.2 The Quote Notation .................... 22
    2.3.5.3 An Interplay between Mathematical
          Notation and Specification Languages .. 22
2.4 Space ................................................. 22
  2.4.1 Space Motivated Philosophically .......... 22
  2.4.2 The Spatial Value ................................ 23
  2.4.3 Spatial Observers ................................ 23
  2.4.4 Spatial Attributes ............................. 23
  2.4.5 Mathematical Models of Space .............. 23
    2.4.5.1 Metric Spaces ............................ 24
2.5 Time ................................................ 24
  2.5.1 Time Motivated Philosophically .......... 25
  2.5.2 Time Values .................................... 25
  2.5.3 Temporal Observers ......................... 26
  2.5.4 J. van Benthem ................................ 26
  2.5.5 Wayne D. Blizard: A Theory of
      Time–Space .................................... 27
  2.5.6 “Soft” and “Hard” Real-time ............... 28
  2.5.7 Soft Temporalities ............................ 28
  2.5.8 Hard Temporalities ............................ 28
  2.5.9 Soft and Hard Real-time ..................... 29
  2.5.10 Temporal Logics ............................. 30
    2.5.10.1 The Issues ............................... 30
    2.5.10.2 A. N. Prior’s Tense Logics ............ 30
    2.5.10.3 The Duration Calculus ................ 31
      2.5.10.3.1 A Function & Safety Example ....... 31
      2.5.10.3.2 The Syntax ........................... 31
        2.5.10.3.2.1 Simple Expressions: ............ 32
        2.5.10.3.2.2 State Expressions and
          Assertions: .................................. 32
        2.5.10.3.2.3 Durations and Duration Terms: ... 32
        2.5.10.3.2.4 Duration Formulas: .............. 32
        2.5.10.3.2.5 Common Duration Formula ....... 32
          Abbreviations: ................................ 32
      2.5.10.3.3 Discussion: From Domains to
        Designs ......................................... 33
2.6 Spatial and Temporal Modelling .................... 33
2.7 Matter .............................................. 33
2.8 Identity and Mereology ............................ 33
  2.8.1 Identity ......................................... 34
  2.8.2 Mereology: Philosophy and Logic .......... 34
    2.8.2.1 Mereology Understood Spatially ....... 34
    2.8.2.2 Our Informal Understanding of
      Mereology ....................................... 35
2.9 A Foundation ...................................... 35
Part II DOMAINS

Chapter 3–7 Overview
A Theory, not The Theory
On Learning a Theory and On Learning a Method
Towards a Theory of Domain Analysis & Description
A Method for Domain Analysis & Description

3 DOMAINS – A Taxonomy

External Qualities
3.1 Overview of this Chapter
3.2 Domains
3.3 Universe of Discourse
3.4 External Qualities
3.5 Entities
3.5.1 A Linnean, Binomial Taxonomy
3.5.2 A Cursory Overview
3.5.3 Summary
3.5.3.1 Space and Time: Whither Entities?

3.6 Endurants and Perdurants
3.6.1 Endurants
3.6.2 Perdurants

3.7 Endurants: Discrete and Continuous
3.7.1 Discrete Endurants
3.7.2 Continuous Endurants: Materials

3.8 Discrete Endurants: Physical Parts, Structures and Living Species
3.8.1 Compound Endurants, “Roots” and “Siblings”
3.8.2 Physical Parts
3.8.3 Structures
3.8.4 Living Species
6: Analysis Prompt: is physical part
7: Analysis Prompt: is-structure
8: Analysis Prompt: is living species

3.9 Physical Parts: Natural Parts and Artefacts
3.9.1 Natural Parts
3.9.2 Artefacts – Man-made Parts
9: Analysis Prompt: is-natural-part
10: Analysis Prompt: is-artefact

3.10 Physical Parts: Structures
3.10.1 General
3.10.2 Compound Structures
11: Analysis Prompt: is-compound-structure
12: Analysis Prompt: is-set-structure

3.11 Physical Parts: Living Species – Plants and Animals
3.11.1 Plants
13: Analysis Prompt: is plant
3.11.2 Animals
14: Analysis Prompt: is animal
3.11.2.1 Humans
15: Analysis Prompt: is human

3.12 Continuous Endurants: Materials
3.13 Natural Parts and Artefacts: Atomic, Composite and Conjoins
3.13.1 Atomic and Compound Parts
3.13.2 Atomic Parts
3.13.3 Composite Parts
3.13.4 Conjoins
3.13.5 Part-Materials Conjoins
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.13.6</td>
<td>Material-Parts Conjoins</td>
<td>58</td>
</tr>
<tr>
<td>3.13.7</td>
<td>Part-Parts Conjoins</td>
<td>59</td>
</tr>
<tr>
<td>3.14</td>
<td>On Discovering Endurant Sorts</td>
<td>60</td>
</tr>
<tr>
<td>3.14.1</td>
<td>On Discovering Man-made Endurants</td>
<td>60</td>
</tr>
<tr>
<td>3.14.2</td>
<td>On Discovering Natural Endurants</td>
<td>60</td>
</tr>
<tr>
<td>3.15</td>
<td>States</td>
<td>61</td>
</tr>
<tr>
<td>3.16</td>
<td>A Review of the Ontology of Endurants</td>
<td>62</td>
</tr>
<tr>
<td>3.17</td>
<td>Endurant Analysis Function Prompts</td>
<td>63</td>
</tr>
<tr>
<td>3.18</td>
<td>Endurant Describers</td>
<td>65</td>
</tr>
<tr>
<td>3.18.1</td>
<td>Describing Composite Parts</td>
<td>66</td>
</tr>
<tr>
<td>3.18.2</td>
<td>Describing Compound Structures</td>
<td>67</td>
</tr>
<tr>
<td>3.18.3</td>
<td>Describing Set Structures</td>
<td>67</td>
</tr>
<tr>
<td>3.18.4</td>
<td>Describing Conjoins</td>
<td>69</td>
</tr>
<tr>
<td>3.18.4.1</td>
<td>Describe Material-Parts Sorts</td>
<td>70</td>
</tr>
<tr>
<td>3.18.4.2</td>
<td>Describe Part-Materials Sorts</td>
<td>71</td>
</tr>
<tr>
<td>3.18.4.3</td>
<td>Describe Part-Parts Sorts</td>
<td>72</td>
</tr>
<tr>
<td>3.18.5</td>
<td>Describing Material</td>
<td>73</td>
</tr>
<tr>
<td>3.19</td>
<td>On Endurant Sorts</td>
<td>73</td>
</tr>
<tr>
<td>3.19.1</td>
<td>Derivation Chains</td>
<td>73</td>
</tr>
<tr>
<td>3.19.2</td>
<td>No Recursive Derivations</td>
<td>73</td>
</tr>
<tr>
<td>3.19.3</td>
<td>Names of Part Sorts and Types</td>
<td>74</td>
</tr>
<tr>
<td>3.20</td>
<td>A Domain Discovery Process, I</td>
<td>74</td>
</tr>
<tr>
<td>3.20.1</td>
<td>A Domain Discovery Notice Board</td>
<td>74</td>
</tr>
<tr>
<td>3.20.2</td>
<td>An Endurant External Qualities Discovery Process</td>
<td>75</td>
</tr>
<tr>
<td>3.20.3</td>
<td>An Assumption</td>
<td>75</td>
</tr>
<tr>
<td>3.21</td>
<td>Formal Concept Analysis</td>
<td>76</td>
</tr>
<tr>
<td>3.21.1</td>
<td>A Formalisation</td>
<td>76</td>
</tr>
<tr>
<td>3.21.2</td>
<td>Types Are Formal Concepts</td>
<td>77</td>
</tr>
<tr>
<td>3.21.3</td>
<td>Practicalities</td>
<td>77</td>
</tr>
<tr>
<td>3.21.4</td>
<td>Formal Concepts: A Wider Implication</td>
<td>77</td>
</tr>
<tr>
<td>3.22</td>
<td>Summary</td>
<td>77</td>
</tr>
<tr>
<td>3.22.1</td>
<td>The Description Schemas</td>
<td>77</td>
</tr>
<tr>
<td>3.22.2</td>
<td>Modelling Choices</td>
<td>78</td>
</tr>
<tr>
<td>3.22.3</td>
<td>Method Principles, Techniques and Tools</td>
<td>78</td>
</tr>
<tr>
<td>3.22.3.1</td>
<td>Principles of External Qualities</td>
<td>78</td>
</tr>
<tr>
<td>3.22.3.2</td>
<td>Techniques of External Qualities</td>
<td>78</td>
</tr>
<tr>
<td>3.22.3.3</td>
<td>Tools of External Qualities</td>
<td>78</td>
</tr>
<tr>
<td>3.22.3.3.1</td>
<td>Tools: Review of the Analysis Calculus</td>
<td>79</td>
</tr>
<tr>
<td>3.22.3.3.2</td>
<td>Tools: Review of the Description Calculus</td>
<td>79</td>
</tr>
<tr>
<td>3.22.4</td>
<td>How Much or How Little Do We Analyse and Describe?</td>
<td>80</td>
</tr>
<tr>
<td>3.23</td>
<td>Bibliographical Notes</td>
<td>80</td>
</tr>
<tr>
<td>3.24</td>
<td>Exercise Problems</td>
<td>80</td>
</tr>
<tr>
<td>3.24.1</td>
<td>Research Problems</td>
<td>80</td>
</tr>
<tr>
<td>3.24.2</td>
<td>A Student Exercise</td>
<td>80</td>
</tr>
<tr>
<td>3.24.3</td>
<td>Term Projects</td>
<td>81</td>
</tr>
<tr>
<td>4</td>
<td>DOMAINS – Towards a Statics Ontology</td>
<td>85</td>
</tr>
<tr>
<td>4.1</td>
<td>Overview of this Chapter</td>
<td>85</td>
</tr>
<tr>
<td>4.2</td>
<td>Unique Identifiers</td>
<td>85</td>
</tr>
<tr>
<td>4.2.1</td>
<td>On Uniqueness of Endurants</td>
<td>86</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Uniqueness Modelling Tools</td>
<td>86</td>
</tr>
<tr>
<td>4.2.3</td>
<td>All Unique Identifiers of a Domain</td>
<td>87</td>
</tr>
<tr>
<td>4.2.4</td>
<td>Unique Identifier Constants</td>
<td>87</td>
</tr>
<tr>
<td>4.2.5</td>
<td>A Domain Law: Uniqueness of Endurant Identifiers</td>
<td>88</td>
</tr>
<tr>
<td>4.3</td>
<td>Mereology</td>
<td>89</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Endurant Relations</td>
<td>89</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Mereology Modelling Tools</td>
<td>90</td>
</tr>
<tr>
<td>4.3.2.1</td>
<td>Invariance of Mereologies</td>
<td>91</td>
</tr>
</tbody>
</table>
4.3.2.2 Deductions made from Mereologies ................................................................. 92
4.3.3 Formulation of Mereologies ............................................................................. 92
4.3.4 Fixed and Varying Mereologies .................................................................. 92
4.3.5 No Materials Mereology ............................................................................... 92
4.3.6 Some Modelling Observations ..................................................................... 93
4.3.7 Conjoin Mereologies ..................................................................................... 93
4.4 Attributes ........................................................................................................... 94
4.4.1 Inseparability of Attributes from Parts and Materials .................................. 94
4.4.2 Attribute Modelling Tools ............................................................................. 95
  4.4.2.1 Attribute Quality and Attribute Value ......................................................... 95
  4.4.2.2 Attribute Types and Functions ................................................................ 95
  4.4.2.3 Attribute Categories ................................................................................ 96
4.4.3 A Discourse on Attribute Kinds .................................................................... 101
  4.4.3.1 A Discussion ............................................................................................ 102
  4.4.3.2 A Preliminary Conclusion ..................................................................... 103
4.4.4 Physics Attributes ........................................................................................ 103
  4.4.4.1 SI: The International System of Quantities ................................................ 104
  4.4.4.2 Units are Indivisible .................................................................................. 105
  4.4.4.3 Chemical Elements .................................................................................. 105
4.4.5 Presentation of Physical Attributes ............................................................... 106
4.4.6 The Care and Feeding of Physical Attributes .............................................. 107
4.4.7 Artefactual Attributes ................................................................................... 107
  4.4.7.1 Examples of Artefactual Attributes .......................................................... 107
4.4.8 Conjoin Attributes ....................................................................................... 108
  4.4.8.1 Conjoin Attribute Categories ................................................................. 108
  4.4.8.2 Conjoin Attribute Assignments ............................................................... 109
4.4.9 Fuzzy Attributes ........................................................................................... 109
  4.4.9.1 Fuzzy Sets and Fuzzy Logic .................................................................... 110
  4.4.9.2 Fuzzy Attribute Types ............................................................................. 110
  4.4.9.3 Fuzzy Attribute Values ........................................................................... 110
  4.4.9.4 Fuzzy Reasoning ..................................................................................... 110
  4.4.9.5 Fuzziness: A Possible Research Topic ? .................................................. 110
4.5 Intentionality ...................................................................................................... 110
  4.5.1 Issues Leading Up to Intentionality ........................................................... 111
  4.5.2 Artefacts ....................................................................................................... 113
  4.5.3 Assignment of Attributes .......................................................................... 113
    4.5.3.1 Artefacts, including Man-made Materials: ........................................... 114
  4.5.4 Galois Connections ..................................................................................... 114
    4.5.4.1 Galois Theory: An Ultra-brief Characterisation .................................... 114
    4.5.4.2 Galois Connections and Intentionality .................................................. 115
    4.5.4.3 Galois Connections and Intentionality: A Possible Research Topic ? .... 115
4.6 Systems Modelling .............................................................................................. 115
  4.6.1 General ......................................................................................................... 115
  4.6.2 Passively Mobile Endurants ....................................................................... 115
4.7 Discussion of Endurants ..................................................................................... 116
4.8 A Domain Discovery Process, II ...................................................................... 116
4.9 Domain Description Laws ............................................................................... 117
4.10 Summary .......................................................................................................... 117
  4.10.1 The Description Schemas .......................................................................... 117
  4.10.2 Modelling Choices ..................................................................................... 118
  4.10.3 Method Principles, Techniques and Tools ............................................... 118
    4.10.3.1 Principles of Internal Qualities ........................................................... 118
    4.10.3.2 Techniques of Internal Qualities ........................................................ 118
    4.10.3.3 Tools .................................................................................................... 119
      4.10.3.3.1 Summary of The Internal Qualities Analysis Calculus .................. 119
      4.10.3.3.2 Summary of The Internal Qualities Description Calculus .......... 119
Contents

7 DOMAIN FACETS

7.1 Introduction

7.1.1 Facets of Domains

7.1.2 Relation to Previous Work

7.1.3 Structure of Chapter

7.2 Intrinsic

7.2.1 Conceptual Analysis

7.2.2 Requirements

7.2.3 On Modeling Intrinsic

7.3 Support Technologies

7.3.1 Conceptual Analysis

7.3.2 Requirements

7.3.3 On Modeling Support Technologies

7.4 Rules & Regulations

7.4.1 Conceptual Analysis

7.4.2 Requirements

7.4.3 On Modeling Rules and Regulations

7.5 Scripts

7.5.1 Conceptual Analysis

7.5.2 Requirements

7.5.3 On Modeling Scripts

7.6 License Languages

7.6.1 Conceptual Analysis

7.6.1.1 The Settings

7.6.1.2 On Licenses

7.6.1.3 Permissions and Obligations

7.6.2 The Pragmatics

7.6.2.1 Digital Media

7.6.2.1.1 Operations on Digital Works

7.6.2.1.2 License Agreement and Obligation

7.6.2.1.3 Two Assumptions
Part III REQUIREMENTS
8 REQUIREMENTS ................................................................. 197
  8.1 Introduction .................................................................. 197
    8.1.1 The Contribution of this Chapter .............................. 197
    8.1.2 Some Comments .................................................... 197
    8.1.3 Structure of Chapter .............................................. 198
  8.2 An Example Domain: Transport ..................................... 198
    8.2.1 Endurants ........................................................... 198
    8.2.2 Domain, Net, Fleet and Monitor .............................. 198
      8.2.2.1 Hubs and Links .............................................. 199
      8.2.2.2 Unique Identifiers ......................................... 199
      8.2.2.3 Mereology .................................................... 200
      8.2.2.4 Attributes, I ................................................... 200
    8.2.3 Perdurants .......................................................... 203
      8.2.3.1 Hub Insertion Action ....................................... 204
      8.2.3.2 Link Disappearance Event ............................... 204
      8.2.3.3 Road Traffic .................................................. 204
        8.2.3.3.1 Global Values: ......................................... 204
        8.2.3.3.2 Channels: ............................................. 205
        8.2.3.3.3 Behaviour Signatures: .............................. 205
        8.2.3.3.4 The Road Traffic System Behaviour: .......... 206
    8.2.4 Domain Facets ..................................................... 207
  8.3 Requirements ............................................................ 208
    8.3.1 Some Requirements Aspects .................................... 209
      8.3.1.1 Requirements Sketches ................................. 209
        8.3.1.1.1 Problem, Solution and Objective Sketch .......... 209
        8.3.1.1.2 Systems Requirements ................................ 210
        8.3.1.1.3 User and External Equipment Requirements .... 210
      8.3.1.2 The Narrative and Formal Requirements Stage .... 211
        8.3.1.2.1 Assumption and Design Requirements .......... 211
    8.3.2 The Three Phases of Requirements Engineering .......... 211
    8.3.3 Order of Presentation of Requirements Prescriptions .. 211
    8.3.4 Design Requirements and Design Assumptions ......... 211
    8.3.5 Derived Requirements .......................................... 212
  8.4 Domain Requirements .................................................. 212
    8.4.1 Domain Projection ................................................ 213
      8.4.1.1 Domain Projection — Narrative .......................... 213
      8.4.1.2 Domain Projection — Formalisation ................... 213
      8.4.1.3 Discussion .................................................. 216
    8.4.2 Domain Instantiation ............................................ 216
      8.4.2.1 Domain Instantiation ..................................... 216
      8.4.2.2 Domain Instantiation — Abstraction .................. 219
      8.4.2.3 Discussion .................................................. 219
    8.4.3 Domain Determination .......................................... 219
      8.4.3.1 Domain Determination: Example ....................... 220
      8.4.3.2 Discussion .................................................. 221
    8.4.4 Domain Extension ............................................... 221
      8.4.4.1 Endurant Extensions ....................................... 221
        8.4.4.1.1 [a] Vehicle Extension: ............................. 222
        8.4.4.1.2 [b] Road Pricing Calculator: Basic Sort and Unique Identifier: ........................................ 222
        8.4.4.1.3 [c] Vehicle to Road Pricing Calculator Channel: .................................................. 222
        8.4.4.1.4 [d] Toll-gate Sorts and Dynamic Types: .......... 223
        8.4.4.1.5 [e] Toll-gate to Calculator Channels: ............ 224
        8.4.4.1.6 [f] Road Pricing Calculator Attributes: ........... 225
        8.4.4.1.7 [g] “Total” System State: ........................... 225
        8.4.4.1.8 [h] “Total” System Behaviour: ..................... 226
      8.4.4.2 Discussion .................................................... 228
    8.4.5 Requirements Fitting .............................................. 228
Chapter 6: Perdurants

© July 6, 2020, 10:03, A Foundation for Software Development

Part V APPENDICES

A A PIPELINES DOMAIN: ENDURANTS ........................................... 281
A.1 Parts and Material ............................................................ 281
A.1.1 Flow Net Parts ............................................................ 281
A.1.1.1 Narrative ............................................................... 281
A.1.1.2 Formalisation ......................................................... 282
A.1.2 Pipeline States ........................................................... 282
A.1.2.1 Narrative ............................................................... 282
A.1.2.2 Formalisation ......................................................... 282
A.2 Unique Identifiers ........................................................... 282
A.3 Mereologies ................................................................. 283
A.3.1 The Pipeline Unit Mereology ........................................... 283
A.3.2 Partial Wellformedness of Pipelines, 0 ............................... 283
A.3.3 Partial Well-formedness of Pipelines, 1 .............................. 283
A.3.3.1 Shared Connectors ................................................ 284
A.3.3.2 Routes ................................................................. 284
A.3.3.3 Wellformed Routes ................................................. 284
A.4 Attributes ....................................................................... 285
A.4.1 Geometric Unit Attributes ............................................. 285
A.4.2 Spatial Unit Attributes .................................................. 285
A.4.3 Unit Action Attributes .................................................. 286
A.4.4 Flow Attributes .......................................................... 286
A.4.4.1 Flows and Leaks ...................................................... 286
A.4.4.2 Intra Unit Flow and Leak Law ................................... 287
A.4.4.3 Inter Unit Flow and Leak Law ................................... 287

Chapter 5: Transcendentality .................................................. 325

Chapter 4: Internal Qualities ................................................... 325

Chapter 3: External Qualities ................................................... 325

Chapter 2: Requirements ......................................................... 325

Chapter 1: Conclusions ................................................................ 325

APPENDICES ............................................................................... 325

INDEX ....................................................................................... 325
# Contents

XXII

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.4.2</td>
<td>Free and Bound Variables</td>
<td>317</td>
</tr>
<tr>
<td></td>
<td>D.4.2.0.1 Free and Bound Variables:</td>
<td>317</td>
</tr>
<tr>
<td>D.4.3</td>
<td>Substitution</td>
<td>317</td>
</tr>
<tr>
<td></td>
<td>D.4.3.0.1 Substitution:</td>
<td>318</td>
</tr>
<tr>
<td>D.4.4</td>
<td>α-Renaming and β-Reduction</td>
<td>318</td>
</tr>
<tr>
<td></td>
<td>D.4.4.0.1 α and β Conversions:</td>
<td>318</td>
</tr>
<tr>
<td>D.4.5</td>
<td>Function Signatures</td>
<td>318</td>
</tr>
<tr>
<td></td>
<td>D.4.5.0.1 Sorts and Function Signatures:</td>
<td>318</td>
</tr>
<tr>
<td>D.4.6</td>
<td>Function Definitions</td>
<td>318</td>
</tr>
<tr>
<td></td>
<td>D.4.6.0.1 Explicit Function Definitions:</td>
<td>319</td>
</tr>
<tr>
<td></td>
<td>D.4.6.0.2 Implicit Function Definitions:</td>
<td>319</td>
</tr>
<tr>
<td>D.5</td>
<td>Other Applicative Expressions</td>
<td>319</td>
</tr>
<tr>
<td>D.5.1</td>
<td>Simple let Expressions</td>
<td>319</td>
</tr>
<tr>
<td></td>
<td>D.5.1.0.1 Let Expressions:</td>
<td>319</td>
</tr>
<tr>
<td>D.5.2</td>
<td>Recursive let Expressions</td>
<td>319</td>
</tr>
<tr>
<td></td>
<td>D.5.2.0.1 Recursive let Expressions:</td>
<td>320</td>
</tr>
<tr>
<td>D.5.3</td>
<td>Predicative let Expressions</td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>D.5.3.0.1 Predicative let Expressions:</td>
<td>320</td>
</tr>
<tr>
<td>D.5.4</td>
<td>Pattern and “Wild Card” let Expressions</td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>D.5.4.0.1 Patterns:</td>
<td>320</td>
</tr>
<tr>
<td>D.5.5</td>
<td>Conditionals</td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>D.5.5.0.1 Conditionals:</td>
<td>320</td>
</tr>
<tr>
<td>D.5.6</td>
<td>Operator/Operand Expressions</td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>D.5.6.0.1 Operator/Operand Expressions:</td>
<td>320</td>
</tr>
<tr>
<td>D.6</td>
<td>Imperative Constructs</td>
<td>321</td>
</tr>
<tr>
<td>D.6.1</td>
<td>Statements and State Changes</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>D.6.1.0.1 Statements and State Change:</td>
<td>321</td>
</tr>
<tr>
<td>D.6.2</td>
<td>Variables and Assignment</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>D.6.2.0.1 Variables and Assignment:</td>
<td>321</td>
</tr>
<tr>
<td>D.6.3</td>
<td>Statement Sequences and skip</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>D.6.3.0.1 Statement Sequences and skip:</td>
<td>321</td>
</tr>
<tr>
<td>D.6.4</td>
<td>Imperative Conditionals</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>D.6.4.0.1 Imperative Conditionals:</td>
<td>321</td>
</tr>
<tr>
<td>D.6.5</td>
<td>Iterative Conditionals</td>
<td>321</td>
</tr>
<tr>
<td></td>
<td>D.6.5.0.1 Iterative Conditionals:</td>
<td>321</td>
</tr>
<tr>
<td>D.6.6</td>
<td>Iterative Sequencing</td>
<td>322</td>
</tr>
<tr>
<td></td>
<td>D.6.6.0.1 Iterative Sequencing:</td>
<td>322</td>
</tr>
<tr>
<td>D.7</td>
<td>Process Constructs</td>
<td>322</td>
</tr>
<tr>
<td>D.7.1</td>
<td>Process Channels</td>
<td>322</td>
</tr>
<tr>
<td></td>
<td>D.7.1.0.1 Process Channels:</td>
<td>322</td>
</tr>
<tr>
<td>D.7.2</td>
<td>Process Composition</td>
<td>322</td>
</tr>
<tr>
<td></td>
<td>D.7.2.0.1 Process Composition:</td>
<td>322</td>
</tr>
<tr>
<td>D.7.3</td>
<td>Input/Output Events</td>
<td>322</td>
</tr>
<tr>
<td></td>
<td>D.7.3.0.1 Input/Output Events:</td>
<td>322</td>
</tr>
<tr>
<td>D.7.4</td>
<td>Process Definitions</td>
<td>323</td>
</tr>
<tr>
<td></td>
<td>D.7.4.0.1 Process Definitions:</td>
<td>323</td>
</tr>
<tr>
<td>D.8</td>
<td>Simple RSL Specifications</td>
<td>323</td>
</tr>
<tr>
<td></td>
<td>D.8.0.0.1 Simple RSL Specifications:</td>
<td>323</td>
</tr>
<tr>
<td>D.9</td>
<td>RSL Module Specifications</td>
<td>323</td>
</tr>
<tr>
<td>D.9.1</td>
<td>Modules</td>
<td>323</td>
</tr>
<tr>
<td>D.9.2</td>
<td>Schemes</td>
<td>324</td>
</tr>
<tr>
<td>D.9.3</td>
<td>Module Extension</td>
<td>324</td>
</tr>
</tbody>
</table>
## INDEXES

- **E.1 Definitions**: 325
- **E.2 Examples**: 328
- **E.3 Method Hints**: 330
- **E.4 Analysis Predicate Prompts**: 330
- **E.5 Analysis Function Prompts**: 331
- **E.6 Attribute Categories**: 331
- **E.7 Perdurant Calculations**: 331
- **E.8 Description Prompts**: 331
- **E.9 Endurant to Perdurant Translation Schemas**: 332
- **E.10 RSL Symbols**: 332
The first three chapters: 0, 1 and 2, provide a personal, intellectual introduction to the field of software engineering.

In Chapter 0 I introduce “my” Concepts. These are briefly characterised. These characterisations can be found in ordinary dictionaries and on Wikipedia. It is their juxtaposition, here in the beginning of this monograph, that, to me is significant and personal. They have formed and form a terminological foundation upon which I have built in the last 40 or more years.

In Chapter 1 I summarize essential aspects of Kai Sørlanders Philosophy. Kai Sørlanders Philosophy, such as I use it, is covered in four of his monographs: [345, 346, 347, 348]. It is thought that this introduction of a philosophical basis for the computer & computing sciences is novel!

In Chapter 2 we bring three fundamental concepts: Space, Time and Matter together. They are all inherent in Kai Sørlander’s Philosophy. I then explore aspects of Identity and Mereology – and refer to the published [76].

Chapter 0 is basically “armchair reading”! Chapters 1 and 2 require a rather more serious study!

---

5 I spell that with a capital P in order to name a specific philosophy
This monograph introduces a rather large number of new concepts. In a conventional software engineering setting, and, as in this case, in a technical/scientific monograph, such an introduction is unusual. I present this chapter because of the large number of new concepts. In order for the reader to find the way around, that reader must be made aware of the background concepts that underlie my treatment of a new branch of software engineering, the domain science and engineering.

0.1 A General Vocabulary

1 Abstraction:

Conception, my boy, fundamental brain-work, is what makes the difference in all art
D.G. Rossetti¹: letter to H. Caine²

Abstraction is a tool, used by the human mind, and to be applied in the process of describing (understanding) complex phenomena.

Abstraction is the most powerful such tool available to the human intellect.

Science proceeds by simplifying reality. The first step in simplification is abstraction. Abstraction (in the context of science) means leaving out of account all those empirical data which do not fit the particular, conceptual framework within which science at the moment happens to be working.

Abstraction (in the process of specification) arises from a conscious decision to advocate certain desired objects, situations and processes as being fundamental; by exposing, in a first, or higher, level of description, their similarities and — at that level — ignoring possible differences.

[From the opening paragraphs of [236, C.A.R. Hoare Notes on Data Structuring]]³

2 Computer: A computer is a collection of hardware and software, that is, a machine that can be instructed to carry out sequences of arithmetic or logical operations automatically via computer programming [Wikipedia].

3 Computer Science: is the study and knowledge of the abstract phenomena that “occur” within computers [DB].

As such computer science includes theory of computation, automata theory, formal language theory, algorithmic complexity theory, probabilistic computation, quantum computation, cryptography, machine learning and computational biology.

¹ Dante Gabrielli Rosetti, 1828–1882, English poet, illustrator, painter and translator
² T. Hall Caine, 1853–1931, British novelist, dramatist, short story writer, poet and critic.
³ We shall bring another quote of Tony Hoare as the last proper text of this monograph, see Sect. 10.8 on Page 259.
4 **Computing Science:** is the study and knowledge of how to construct “those things” that “occur” within computers [DB].

As such computing science embodies algorithm and data structure design, functional-, logic-, imperative- and parallel programming; code testing, model checking and specification proofs. Much of this can be pursued using formal methods (see Item 9).

5 **Domain Engineering:** is the engineering of domain descriptions based on the engineering of domain analyses [DB].

Chapters 3–7 covers domain engineering.

We shall later, in Chapter 10, summarise the “benefits” of domain engineering. Suffice it here to say that basing software development on domain analysis & description shall help secure that the eventually emerging software meets customer expectations.

6 **Domain Requirements:** are those requirements which can be expressed solely in terms of domain concepts.

7 **Engineering:** is the use of scientific principles to design and build machines, structures, and other items, including bridges, tunnels, roads, vehicles, and buildings [Wikipedia].

The engineer walks the bridge between science and technology: analysing man-made devices for their possible scientific properties and constructing technology based on scientific insight.

We refer to [ι 5.π 4], [ι 27.π 5], [ι 35.π 6].

8 **Epistemology:** is the branch of philosophy concerned with the theory of knowledge – and is the study of the nature of knowledge, justification, and the rationality of belief [Wikipedia].

9 **Formal Method:** By a formal method we shall here understand a method whose techniques and tools can be understood mathematically.

For formal domain, requirements or software engineering methods formality means the following:

- There is a set, one or more, specification languages – say for domain descriptions, requirements prescriptions, software specifications, and software coding, i.e., programming languages. 4
- These are all to be formal, that is, to have a formal syntax, a formal semantics, and a formal, typically Mathematical Logic proof system.
- Some of the techniques and tools must be supported by a mathematical understanding.

10 **Hardware:** The physical components of a computer: electronics, mechanics, etc. [Wikipedia].

11 **Interface Requirements:** are those requirements which can be expressed in a combination of both domain and machine concepts. They do so because certain entities, whether endurants or perdurants, are shared between the domain and the machine.

12 **Language:** By language we shall, with [Wikipedia], mean a structured system of communication.

Language, in a broader sense, is the method of communication that involves the use of – particularly human– languages. The ‘structured system’ that we refer to has come to be known as Syntax, Semantics and Pragmatics. We refer to [ι 37.π 7], [ι 31.π 6], and [ι 25.π 5].

13 **Linguistics:** By linguistics we shall mean the scientific study of language.

14 **Machine:** By a machine we shall understand a combination of software and hardware.

15 **Machine Requirements:** are those requirements which can be expressed solely in terms of machine concepts.

16 **Mathematics:** By mathematics we shall here understand a such human endeavours that makes precise certain facets of language, [ι 12.π 4] whether natural or ‘constructed’ (as for mathematical notation), and out of those endeavours, i.e., mathematical constructions, also called theories, build further abstractions. We refer to Sects. 2.2 on Page 17–2.3 on Page 17.

---

4 Most formal specification languages are textual, but graphical languages like Petri nets [333], Message Sequence Charts[249], Statecharts [199], Live Sequence Charts [200], etc., are also formal.

5 By [ι 37.π 7] we mean to refer to item 37 page 7
17 **Metaphysics**: is the branch of philosophy that examines the fundamental nature of reality, including the relationship between mind and matter, between substance and attribute, and between potentiality and actuality [269] [Wikipedia].
In this monograph we stay clear of metaphysics.

18 **Mereology**: is the theory of parthood relations: of the relations of part to whole and the relations of part to part within a whole [358, 344, 115].
The term ‘mereology’ is accredited to the Polish mathematician, philosopher and logician Stanislaw Leśniewski (1886–1939).

19 **Method**: By a method we shall understand a set of **principles** for selecting and applying a set of **techniques** using a set of **tools** in order to construct an artefact [DB].
We shall in this primer focus on a method for pursuing domain analysis and for constructing domain descriptions. Key chapters will summarise some methodological aspects of their content.

20 **Methodology**: is the comparative study and knowledge of methods [DB].
[The two terms: ’method’ and ’methodology’ are often confused, including used interchangeably.]

21 **Model**: A mathematical model is a description of a system using mathematical concepts and language.
We shall include descriptions\(^6\), prescriptions\(^7\) and specifications\(^8\) using formal languages as presenting models.

22 **Modelling**: Modelling is the act of creating models, which include discrete mathematical structures (sets, Cartesians, lists, maps, etc.), and are logical theories represented as algebras. That is, any given RSL text denotes a set of models, and each model is an algebra, i.e., a set of named values and a set of named operations on these. Modelling is the engineering activity of establishing, analysing and using such structures and theories. Our models are established with the intention that they “model” “something else” other than just being the mathematical structure or theory itself. That “something else” is, in our case, some part of a reality\(^9\), or of a construed such reality, or of requirements to the, or a reality\(^10\), or of actual software\(^11\).

23 **Ontology**: is the branch of metaphysics dealing with the nature of being; a set of concepts and categories in a subject area or domain that shows their properties and the relations between them [108, 109] [Wikipedia].
In this monograph we shall, indeed, focus much on the ontology of domains. See, f.ex., Chapter 4.

24 **Philosophy**: is the study of general and fundamental questions about existence, knowledge, values, reason, mind, and language. Such questions are often posed as problems to be studied or resolved [Wikipedia].

25 **Pragmatics**: studies the ways in which context contributes to meaning. Pragmatics encompasses speech act theory, conversational implicature, talk in interaction and other approaches to language behavior in philosophy, sociology, linguistics and anthropology [304, 288] [Wikipedia].

26 **Requirements**: By a requirements we understand (cf., [245, IEEE Standard 610.12]): “A condition or capability needed by a user to solve a problem or achieve an objective”
In software development the requirements explain what properties the desired software should have, not how these properties might be attained. In our, the triptych approach, requirements are to be “derived” from domain descriptions.

27 **Requirements Engineering**: is the engineering of constructing requirements [DB].
The aim of requirements engineering is to design the machine. Chapter 8 covers requirements engineering.

28 **Requirements Prescription:** By a requirements prescription we mean a document which outlines the requirements that some software is expected to fulfill.

29 **Requirements Specification:** By a requirements specification we mean the same as a requirements prescription.

30 **Science:** is a systematic enterprise that builds and organizes knowledge in the form of testable explanations and predictions about the universe [Wikipedia].
Science is the intellectual and practical activity encompassing the systematic study of the structure and behaviour of the physical and natural world through observation and experiment.

31 **Semantics:** is the linguistic and philosophical study of meaning in language, programming languages, formal logics, and semiotics. It is concerned with the relationship between signifiers — like words, phrases, signs, and symbols — and what they stand for in reality, their denotation [112] [Wikipedia].

The languages that we shall be concerned with is, on one hand, the language[s] in which we describe domains [as here a variant of RSL, the RAISE Specification Language, extended, as we shall see in Chapters 3–6.] and, on the other hand, the language that emerges as the result of our domain analysis & description: a domain specific language.

There are basically three kinds of semantics, expressed somewhat simplistically:

- **Denotational Semantics** model-theoretically assigns a meaning, a denotation, to each phrase structure, i.e., syntactic category.
- **Axiomatic Semantics** or Mathematical Logic Proof Systems is an approach based on mathematical logic for proving the correctness of specifications.
- **Algebraic Semantics** is a form of axiomatic semantics based on algebraic laws for describing and reasoning about program semantics in a formal manner.

32 **Semiotics:** is the study and knowledge of sign process (semiosis), which is any form of activity, conduct, or any process that involves signs, including the production of meaning [Wikipedia]. A sign is anything that communicates a meaning, that is not the sign itself, to the interpreter of the sign. The meaning can be intentional such as a word uttered with a specific meaning, or unintentional, such as a symptom being a sign of a particular medical condition. Signs can communicate through any of the senses, visual, auditory, tactile, olfactory, or gustatory [Wikipedia].

The study and knowledge of semiotics is often “broken down” into the studies, etc., of syntax, semantics and pragmatics.

33 **Software:** is the set of all the documents that have resulted from a completed software development: domain analysis & description, requirements analysis & prescription, software: software code, software installation manuals, software maintenance manuals, software users guides, development project plans, budget, etc.

34 **Software Design:** is the engineering of constructing software [DB].
Whereas software requirements engineering focus on the logical properties that desired software should attain, software design, besides focusing on achieving these properties correctly, also focus on the properties being achieved efficiently.

35 **Software Engineering:** to us, is then the combination of domain and requirements engineering with software design [DB].
This is my characterisation of software engineering. It is at the basis of this monograph as well as [39, 40, 41].

36 **Software Development:** is then the combination of the development of domain description, requirements prescription and software design [DB].
This is my characterisation of software engineering. It is at the basis of this monograph as well as [39, 40, 41].
0.2 More on Method

We elaborate on issues arising from the concept of ‘method’. These are brought here in some, hopefully meaningful, but not alphabetic order!

41 **Method**: By a method we shall understand a **set of principles** for selecting and applying a **set of techniques** using a **set of tools** in order to construct an artefact [DB].

42 **Principle**: By a **principle** we shall, loosely, understand (i) elemental aspect of a craft or discipline, (ii) foundation, (iii) general law of nature, etc [www.etymonline.com].

43 **Technique**: By a **technique** we shall, loosely, understand (i) formal practical details in artistic, etc., expression, (ii) art, skill, craft in work” [www.etymonline.com]. Classical technique are that of establishing **invariants** and expressing **intentional pull**. See Item 49 on the following page and Item 48 on the next page.

44 **Tool**: By a **tool** we shall, loosely, understand (i) instrument, implement used by a craftsman or laborer, weapon, (ii) that with which one prepares something, etc. [www.etymonline.com].

We shall, at the end of several chapters\(^\text{12}\) summarise the principles, techniques and tools covered by these chapters.

Among basic principles, to be applied across all phases of software development, and hence in all phased of software engineering are those of:

45 **Abstraction**: We refer to Item 1 on Page 3.

46 **Conservative Extension**: An extension of a logical theory is conservative, i.e., conserves, if every theorem expressible in the original theory is also derivable within the original theory [en.wiktionary.org/wiki/conservative_extension].

47 **Divide and Conquer**: In computer science, divide and conquer is an algorithm design paradigm based on multi-branched recursion. A divide-and-conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly [Wikipedia].

But this monograph is not about the exciting field of algorithm design.

Yet, the principle of **divide and conquer** is also very strongly at play here: In the top-down analysis of a domain into what can be described and what is indescribable, of describable entities into endurants and perdurants, of endurants into discrete, conjoins and materials, of discrete into physical parts, structures and living species, and so forth [cf. Fig. 3.1 on Page 44].

---

\(^{12}\) See Sects. 3.22.3 on Page 78, 4.10.3 on Page 118, 6.13.1 on Page 158, 7.9.1 on Page 190, 8.7.1 on Page 238
CONCEPTS

48 **Intentional Pull:** The concept of intentional pull is a wider notion than that of invariant. Here we are not concerned with pre-/post-conditions on operations. Intentional pull is exerted between two or more phenomena of a domain when their relation can be asserted to **always** hold.

49 **Invariants:** The concept of invariants in the context of computing science is most clearly illustrated in connection with the well-formedness of data structures. Invariants then express properties that must hold, i.e., as a **pre-condition**, before any application of an operation to those data structures and shall hold, i.e. as a **post-condition** after any application of an operation to those data structures.

50 **Narration & Formalisation:** To communicate what a domain “is”, one must be able to narrate of what it consists. To understand a domain one must give a formal description of that domain. When we put an ampersand, &, between the two terms we mean to say that they form a whole: not one without the other, either way around! In our domain descriptions we enumerate narrative sentences and ascribe this enumeration to formal expressions.

51 **Nondeterminism:** Non-determinism is a fundamental concept in computer science. It appears in various contexts such as automata theory, algorithms and concurrent computation. …The concept was developed from its inception by Rabin & Scott, Floyd and Dijkstra; as was the interplay between non-determinism and concurrency [Michal Armoni and Mordechai Ben-Ari].

52 **Operational Abstraction** abstract the way in which we express operations on usually representationally abstracted values. In conventional programming we refer to operational abstract as **procedure abstraction**.

53 **Refinement** is a verifiable transformation of an abstract (i.e., high-level) formal specification into a less abstract, we say more concrete (i.e., low-level) specification or an executable program. Step-wise refinement allows the refinement of a program, from a specification, to be done in stages [www.igi-global.com/dictionary].

54 **Representational Abstraction** abstracts the representation of type values, say in the form of just plain **sorts**, or, when concrete types, then in, for example the form of mathematical sets, or maps (i.e., discrete functions, usually from finite definition sets into likewise representationally abstracted ranges), or Cartesians (i.e., groupings of likewise abstracted elements), etc. In conventional programming we refer to representational abstract as **data abstraction**.

55 **Syntax and Semantics:** When we write:

\[
\text{let } a : \text{A in } B(a) \text{ end}
\]

We mean that the [free] \(a\) in the \(B(a)\) clause is bound to the value \(a\) of type \(\text{A}\) in \text{let } a:A.

56 **Syntax Names:** To express that we refer to the syntactic name of a sort or type, \(\text{A}\), we write:

\[
\text{"A"}
\]

That is, “...” is a special, meta-linguistic distributed-fix **quote [unquote]** operator. It is explained in Sect. 2.3.5.2 Page 22.

0.3 **Some More Personal Observations**

- **Informatics:** We understand informatics as a confluence of mathematics, of the computer and computing sciences, of the domain science and engineering as espoused in this monograph, requirements engineering and software design.
- **IT – Information Technology:** We understand information technology as the confluence of nano physics, electronics, computers and communication (hardware), sensors, actuators, etc.
- **Two Universes:** Two diverse universes appear to emerge: **Information Technology** is, to this author, a universe of both material quality and quantity. It is primarily materially characterised, such as I see it, by such terms as bigger, smaller; faster, slower; costly, inexpensive, and environment “friendly”.
Informatics is, to this author, a universe of intellectual quality. As such it is primarily characterised, such as I see it, by such terms as better, more fit for purpose, appropriate, logically correct and meets user expectations.
PHILOSOPHY

In this chapter we cover notions of philosophy that we claim are fundamental to our understanding of domain science and engineering.

We shall base some of our domain analysis decisions on Kai Sørlander’s Philosophy [345, 346, 347, 348]. A main contribution of Kai Sørlander is, on the philosophical basis of the possibility of truth (in contrast to Kant’s possibility of self-awareness), to rationally and transcendentally deduce the absolutely necessary conditions for describing any world. These conditions presume a principle of contradiction and lead to the ability to reason using logical connectives and to handle asymmetry, symmetry and transitivity. Transcendental deductions then lead to space and time, not as priory assumptions, as with Kant, but derived facts of any world. From this basis Kai Sørlander then, by further transcendental deductions, arrive at kinematics, dynamics and the bases for Newton’s Laws.

We build on Sørlander’s basis to argue that the domain analysis & description calculi are necessary and sufficient for the analysis & description of domains and that a number of relations between domain entities can be understood transcendentally and as “variants” of laws of physics, biology, etc.!

1.1 Some Issues of Philosophy

The question is: “what, if anything, is of such necessity, that it could under no circumstances be otherwise?” or “which are the necessary characteristics of any possible world?”. We take it that the necessary characteristics of any domain is equivalent with the conceptual, logical conditions for any possible description of that domain. Sørlander puts forward the thesis of the possibility of truth and then, basing transcendental deductions on indisputable logical relations, arrives at the conceptual, logical conditions for any possible description of any domain.

The starting point, now, in a series of deductions, is that of logic and that we can assert a property, \( P \), and its negation \( \neg P \). These two assertions cannot both be true, that is, that \( P \land \neg P \) cannot be true. So the possibility of truth is a universally valid condition. When we claim that, we also claim the contradiction principle. The implicit meaning theory is this: “in assertions there are mutual dependencies between the meaning of designations and consistency relation between assertions”. When we claim that a philosophy basis is that of the possibility of truth, then we assume that this basis include the contradiction principle and the implicit meaning theory. We shall also refer to the implicit meaning theory as the inescapable meaning assignment.

As an example of what “goes into” the inescapable meaning assignment, we bring, albeit from the world of computer science, that of the description of the stack data type (its endurants and operations).

### Inescapable Meaning Assignment, Narrative

**Example 1** The meaning of designations:

57 Stacks, \( s:S \), have elements, \( e:E \);
58 the empty \( s \) operation takes no arguments and yields a result stack;
59 the is_empty \( s \) operation takes an argument stack and yields a Boolean value result.
60 the stack operation takes two arguments: an element and a stack and yields a result stack.
61 the unstack operation takes an non-empty argument stack and yields a stack result.
62 the top operation takes an non-empty argument stack and yields an element result.
The consistency relations:

63 an empty $S$ stack is_empty, and a stack with at least one element is not;
64 unstacking an argument stack, stack($e, s$), results in the stack $s$; and
65 inquiring the top of a non-empty argument stack, stack($e, s$), yields $e$.

Inescapable Meaning Assignment, Formalisation

Example 2

The meaning of designations:

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$E$, $S$</td>
</tr>
<tr>
<td>2.</td>
<td>empty,$S$: Unit $\rightarrow S$</td>
</tr>
<tr>
<td>3.</td>
<td>is_empty,$S$: $S \rightarrow$ Bool</td>
</tr>
<tr>
<td>4.</td>
<td>stack: $E \times S \rightarrow S$</td>
</tr>
<tr>
<td>5.</td>
<td>unstack: $S \sqty S$</td>
</tr>
</tbody>
</table>

The consistency relations:

6. top: $S \sqty E$

7. is_empty(empty $S()$) = true
7. is_empty(stack($e, s$)) = false
8. unstack(stack($e, s$)) = $s$
9. top(stack($e, s$)) = $e$

1.2 Transcendence

Definition: 1 Transcendental, I: By transcendental we shall understand the philosophical notion: the a priori or intuitive basis of knowledge, independent of experience.

Definition: 2 Transcendental Deduction, I: By a transcendental deduction we shall understand the philosophical notion: a transcendental ‘conversion’ of one kind of knowledge into a seemingly different kind of knowledge.

Transcendental philosophy, with Kant and Sørlander, seeks to find the necessary conditions for experience, recognition and understanding. Transcendental deduction is then the “process”, based on the principle of contradiction and the implicit meaning theory, by means of which – through successive concept definitions – one can deduce a system of base concepts which must be assumed in any possible description of the world. The subsequent developments of the logical connectives, modalities, existence, identity, difference, relations, numbers, space, time and causality, are all transcendental deductions.

We shall return to the notions of transcendence in Chapter 5.

1.3 Overview of The Sørlander Philosophy

In this section we shall give a very terse summary of main elements of Kai Sørlander’s philosophy. We shall primarily base this overview on [348]. It is necessarily a terse summary. What we overview is developed in [348] over some 50 pages. Sørlander’s books [345, 346, 347, 348], relevant to this overview, are all in Danish. Hence the need for this section.

1.3.1 Logical Connectives

1.3.1.1 Negation: $\neg$:

The logical connective, negation ($\neg$), is defined as follows: if assertion $\mathcal{P}$ holds then assertion $\neg\mathcal{P}$ does not hold. That is, the contradiction principle understood as a definition of the concept of negation.
1.3.1.2 Conjunction and Disjunction: $\land$ and $\lor$

Assertion $P \land Q$ holds, i.e., is true, if both $P$ and $Q$ holds. Assertion $P \lor Q$ holds, i.e., is true, if either $P$ or $Q$ or both $P$ and $Q$ holds.

1.3.1.3 Implication: $\Rightarrow$

Assertion $P \Rightarrow Q$ holds, i.e., is true, if the first assertion, $P$, holds, $t$, and the second assertion, $Q$, is not false, $\neg f$. Used in logic is also called material implication.

1.3.2 Towards a Philosophy–basis for Physics and Biology

In a somewhat long series of deductions we shall, based on Sørland’s Philosophy, motivate the laws of Newton and more, not on the basis of empirical observations, but on the basis of transcendental deductions and rational reasoning.

1.3.2.1 Possibility and Necessity

Based on logical implication we can transcendentally define the two modal operators: necessity and possibility.

**Definition:** 3 Necessarily True Assertions: An assertion is necessarily true if its truth follows from the definition of the designations by means of which it is expressed.

**Definition:** 4 Possibly True Assertions: An assertion is possibly true if its negation is not necessary.

1.3.2.2 Empirical Assertions

There can be assertions whose truth value does not only depend on the definition of the designations by means of which they are expressed. Those are assertions whose truth value depend also on the assertions referring to something that exists independently of the designations by means of which they are expressed. We shall call such assertions empirical.

1.3.2.3 Existence of Entities

With Sørlander we shall now argue that there exist many entities in any world: “Entities, in a first step of reasoning, that can be referred to in empirical assertions, do not necessarily exist. It is, however, an empirical fact that they do exist; hence there is a logical necessity that they do not exist. In a second step of reasoning, these entities must exist as a necessary condition for their actually being ascribed the predicates which they must necessarily befit in their capacity of being entities referred to in empirical assertions.”

---

1 Here we need to emphasize that the above quote from Sørlander is one between the type and a value of that type. So the empirical assertions motivate that we speak of the type of an entity. Empirical facts then states that some specific value of that type need not exist. In fact, there is, most likely, an indefinite number of values of the asserted type that do not exist.
1.3.2.4 Identity, Difference and Relations

[348, pp 146] “An entity, referred to by \( A \), is identical to an entity, referred to by \( B \), if \( A \) cannot be ascribed a predicate in-commensurable with a predicate ascribed to \( B \).” That is, if \( A \) and \( B \) cannot be ascribed in-commensurable predicates. [348, pp 146] “Entities \( A \) and \( B \) are different if they can be ascribed in-commensurable predicates.” [348, pp 147] “Identity and difference are relations.” [348, pp 147] “As a consequence identity and difference imply relations. Symmetry and asymmetry are also relations: \( A \) identical to \( B \) is the same as \( B \) identical to \( A \). And \( A \) different from \( B \) is the same as \( B \) different from \( A \). Finally transitivity follows from \( A \) identical to \( B \) and \( B \) identical to \( C \) implies \( A \) identical to \( C \).”

1.3.2.5 Sets

We can, as a consequence of two or more different entities satisfying a same predicate, say \( P \), define the notion of the set of all those entities satisfying \( P \). And, as a consequence of two or more entities, \( e_1, ..., e_j \), all being distinct, therefore implying in-commensurable predicates, \( Q_1, ..., Q_j \), but still satisfying a common predicate, \( P \), we can claim that they all belong to a same set. The predicate \( P \) can be said to type that set. And so forth: following this line of reasoning we can introduce notions of cardinality of sets, finite and infinite sets, existential (\( \exists \)) and universal (\( \forall \)) quantifiers, etc.; and we can in this way transcendently deduce the concept of (positive) numbers, their addition and multiplication; and that such are an indispensable aspect of any domain. We leave it then to mathematics to study number theory.

1.3.2.6 Space and Geometry

**Definition:** 5 Space: [348, pp 154] “The two relations asymmetric and symmetric, by a transcendental deduction, can be given an interpretation: the relation (spatial) direction is asymmetric; and the relation (spatial) distance is symmetric. Direction and distance can be understood as spatial relations. From these relations are derived the relation in-between. Hence we must conclude that primary entities exist in space. Space is therefore an unavoidable characteristic of any possible world.”

[348, pp 155] “Entities, to which reference can be made in simple, empirical assertions, must exist in space; they must be spatial, i.e., have a certain extension in all directions; they must therefore “fill up some space”, have surface and form.” From this, by further reasoning one can develop notions of points, line, surface, etc., i.e., Euclidean as well as non-Euclidean geometry. We refer to Sects. 2.4 on Page 22 and 2.4.4 on Page 23 for more on space.

1.3.2.7 States

We introduce a notion of state. [348, pp 158–159] “Entities may be ascribed predicates which it is not logically necessary that they are ascribed. How can that be possible? Only if we accept that entities may be ascribed predicates which are in-commensurable with predicates that they are actually ascribed.” That is possible, we must conclude, if entities can exist in distinct states. We shall let this notion of state further undefined – till Sect. 3.15.

1.3.2.8 Time and Causality

**Definition:** 6 Time: [348, pp 159] “Two different states must necessarily be ascribed different incompatible predicates. But how can we ensure so? Only if states stand in an asymmetric relation to one another. This state relation is also transitive. So that is an indispensable property of any world. By a transcendental deduction we say that primary entities exist in time. So every possible world must exist in time.”
We refer to Sect. 2.5 on Page 24 for more on time.

So space and time are not phenomena, i.e., are not entities. They are, by transcendental reasoning, aspects of any possible world, hence, of any description of any domain. In a concentrated series [348, 160-163] of logical reasoning and transcendental deductions, Sørlander introduce the concepts of the empirical circumstances under which entities exist, implying non-logical implication between one-and-the-same entity at distinct times, leading to the notions of causal effect and causal implication – all deduced transcendentally. Whereas Kant’s causal implication is transcendentally deduced as necessary for the possibility of self-awareness. Sørlander’s causal implication does not assume possibility of self-awareness. The principle of causality is a necessary condition for assertions being about the same entity at different times.

1.3.2.9 Kinematics

[348, pp 164] “Entities are in both space and time; therefore it must be assumed that they can change their spatial properties; that is, are subject to movement. An entity which changes location is said to move. An entity which does not change location is said to be at rest.” In this way [348] transcendentally introduces the notions of velocity and acceleration, hence kinematics.

1.3.2.10 Dynamics

[348, pp 166] “When combining the causality principle with dynamics we deduce that when an entity changes its state of movement then there must be a cause, and we call that cause a force.” [348, pp 166] “The change of state of entity movement must be proportional to the applied force; an entity not subject to an external force will remain in its state of movement: This is Newton’s 1st Law.”

[348, pp 166] “But to change an entity’s state of movement, by some force, must imply that the entity exerts a certain resistance to that change; the entity must have a mass. Changes in an entity’s state of movement besides being proportional to the external force, must be inverse proportional to its mass. This is Newton’s 2nd Law.”

[348, pp 166-167] “The forces that act upon entities must have as source other entities: entities may collide; and when they collide the forces they exert on each other must be the same but with opposite directions. This is Newton’s 3rd Law.”

[348, pp 167-168] “How can entities be the source of forces? How can they have a mass? Transcendentally it must follow from what we shall refer to as gravitational pull. Across all entities of mass, there is a mutual attraction, Universal Gravitation.” [348, pp 168-169] “Gravitation must, since it has its origin in the individual entities, propagate with a definite velocity; and that velocity must have a limit, a constant of nature, the universal speed limit.”

1.4 From Philosophy to Physics and Biology

Based on logical reasoning and transcendental deductions one can thus derive major aspects of that which must be (assumed to be) in any description of any world, i.e., domain. In our domain description ontology we shall let the notions of discrete endurants (parts) and continuous endurants (non-solids) cover what we have covered so far: they are those entities which satisfy the laws of physics, hence are in space and time. In the next sections we shall make further use of Sørlander’s Philosophy to logically and transcendentally justify the inevitability of living species: plants and animals including, notably, humans, in any description of any domain.

1.4.0.1 Purpose, Life and Evolution

[348, pp 174] “For language and meaning to be possible there must exist entities that are not constrained to just the laws of physics. This is possible if such entities are further subject to a “purpose-causality”
directed at the future. These entities must strive to maintain their own existence.” We shall call such entities **living species**. Living species must maintain and also further develop their form and do so by an exchange of materials with the surroundings, i.e., **metabolism**, with one kind of living species subject only to development, form and **metabolism**, while another kind additionally move **purposefully**. The first we call **plants**, the second **animals**. Animals, consistent with the principle of causality, must possess **sensory organs**, a **motion apparatus**, and **instincts, feelings, promptings** so that what has been sensed, may be responded to [through motion]. The **purpose-directness** of animals must be built into the animals. Biology shows that that is the case. The animal genomes appear to serve the **purpose-directness** of animals. [348, pp 178] “Biology shows that it is so; transcendental deduction that it must be so.”

### 1.4.0.2 Awareness, Learning and Language

[348, pp 180] “Animals, to **learn** from **experience**, must be able to feel **inclination** and **disinclination**, and must be able to remember that it has acted in some way leading to either the feeling of inclination or disinclination. As a consequence, an animal, if when acting in response to sense impression, i, experiences the positive feeling of inclination (desire), then it will respond likewise when again receiving sense impression i, until it is no longer so inclined. If, in contrast, the animal feels the negative feeling of disinclination (dislike), upon sense impression i, then it will avoid responding in this manner when receiving sense impression i.” [348, pp 181] “Awareness is built up from the sense impressions and feelings on the basis of, i.e., from what the individual animal has learned. Different animals can be expected to have different levels of consciousness; and different levels of consciousness assume different biological bases for learning. This is possible, biology tells us, because of there being a central nervous system with building blocks, the neurons, having an inner determination for learning and consciousness.” [348, pp 181–182] “In the mutual interaction between animals of a higher order of consciousness these animals learn to use **signs** developing increasingly complex sign systems, eventually “arriving” at **languages**.” It is thus we single out **humans**. [348, pp 183] “Any human language which can describe reality, must assume the full set of concepts that are prerequisites for any world description.”

### 1.5 Philosophy, Science and the Arts

We quote extensively from [346, Kai Sørlander, 1997].

[346, pp 178] “Philosophy, science and the arts are products of the human mind.”

[346, pp 179] “Philosophy, science and the arts each have their own goals.

- **Philosophers** seek to find the inescapable characteristics of any world.
- **Scientists** seek to determine how the world actually, really is, and our situation in that world.
- **Artists** seek to create objects for experience.

We shall elaborate.” [346, pp 180] “Simplifying, but not without an element of truth, we can relate the three concepts by the **modalities**:

- **philosophy is the necessary**.
- **science is the real**, and
- **art is the possible**.

. . . Here we have, then, a distinction between philosophy and science. . . . From [345] we can conclude the following about the results of philosophy and science. These results must be consistent [with one another]. This is a necessary condition for their being **correct**. . . . The **real** must be a **concrete realisation** of the **necessary**.”
From Kai Sørlander’s Philosophy we can, by logical reasoning, infer space and time as facts. We do not have to presume them, as did Immanuel Kant. In this chapter we shall examine the concepts of space and time, as already introduced in the previous chapter, and introduce the concept of matter as more-or-less ex/implicitly referred to also in Kai Sørlander’s Philosophy.

2.1 Prologue

There are three main elements of this chapter. They are (i) An introduction to notions of language, logic and mathematics. (ii) The main elements on space, time and matter. They were already introduced in Chapter 1. In the first two of these we shall make use of notions of mathematical logic introduced earlier. (iii) Finally a cursory, i.e., an initial view of the notions of identity and mereology – also already introduced in that chapter, Chapter 1. Identity and mereology will be core internal qualities of endurants – and dealt with in Sects. 4.2 and 4.3. Mereology will be further treated in Appendix B.

Space can be logically reasoned to exist. So can time. Matter is implicit in Kai Sørlander’s Philosophy – in that the properties that can be expressed about entities include properties that, by transcendental deduction, entail matter: that which one can see and touch and those which can be "measured", that is, that exist in space and satisfy laws of nature.

2.2 Logic

Definition: 7 Logic: By logic we shall here mean: the kind of reasoning that was shown in Chapter 1. It was based on the possibility of truth, and hence on the necessity of negation, from which was derived the logic operators conjunction and disjunction and, subsequently, as a result of the necessity of existence, identity and multiplicity of entities, the equality operator.

In this monograph we are indeed very much concerned with the logic of domains.

Philosophical logic is the branch of study that concerns questions about reference, predication, identity, truth, quantification, existence, entailment, modality, and necessity. Philosophical logic is includes the application of formal logical techniques to philosophical problems.

2.3 Mathematics

Mathematics has no generally accepted definition\textsuperscript{1}.

By mathematics we shall here, operationally\textsuperscript{2} mean the study and knowledge of algebra, calculus, combinatorics, geometry, graph theory, logic, whence mathematical logic, number theory, probability, set theory, statistics et cetera. – alphabetically listed!

\textsuperscript{1} Mathematics is what mathematicians do!

\textsuperscript{2} – that is, in terms of the names of fields of mathematics
2.3.1 Mathematical Logic

Definition: 8 Mathematical Logic: By mathematical logic we shall here mean: the study and knowledge of set theory, propositions, predicates, first-order logic, definability, model theory, proof theory and recursion theory. — more-or-less arbitrarily listed!

Some basic notions of mathematical logic are: truth values: true, false, chaos³, ¬true, ¬false, true∧false, ¬true∧false, ... ground terms: ¬a, a∧b, a∨b, a⇒b, a≡b, a≡b (variables a, b, ... to range over truth values), propositions: true, false, ¬true, ¬false, true∧false, ¬true∧false, ..., a, b, ..., a∧true, a∧b, ... and predicates: true, false, ¬true, ¬false, true∧false, ¬true∧false, ..., a, b, ..., a∧true, a∧b, ..., ∀x:X∙true, ∀x:X∙x∧..., ∃x:X∙x∧...

Three cornerstones of mathematical logic are: inference rules, axiom systems and proofs.

Definition: 9 Inference Rules: An inference rule consists of a list of one or more premises (predicates) and a conclusion (also a logical term):

\[ p_1, p_2, ..., p_n \vdash c. \]

This expression states that whenever in the course of some logical derivation the given premises, \( (p_1, p_2, ..., p_n) \), have been obtained, the specified conclusion, \( c \), can be taken for granted as well.

Definition: 10 Axioms and Axiom System: An axiom (or a postulate) is a statement that is taken to be true, to serve as a premise or starting point for further reasoning and arguments.

An axiom system is a set of one or more axioms.

We illustrate some axiom systems. Metric Space. Axiom System 1 on Page 24; J. van Benthem’s A Continuum Theory of Time, Axiom System 2 on Page 27; and Wayne D. Blizard’s A Theory of Time-Space, Axiom System 3 on Page 27. These are brought, not because we shall actually ‘use’ them, but to illustrate what axiom systems are.

Definition: 11 Proof: Proof, in logic, is an argument that establishes the validity of a proposition, \( p \). The argument usually requires a sequence of proof steps, \( i \), each, usually, refers to the steps in the argument that represents the premises, a proof rule, and the conclusion, \( c \), which becomes a new step.

Some related concepts of mathematical logic in software engineering are: interpretation, satisfiability, validity and model.

Definition: 12 Interpretation: By an interpretation of a predicate we mean an assignment of a truth value to a predicate where the assignment may entail an assignment of values, in general, to the terms of the predicate.

Definition: 13 Satisfiability: By the satisfiability of a predicate we mean that the predicate is true for some interpretation.

Definition: 14 Validity: By the validity of a predicate we mean that the predicate is true for all interpretations.

Definition: 15 Model: By a model of a predicate (an axiom system) we mean an interpretation for which the predicate (of the axiom system) holds.

³ Yes, our notations, both the mathematical and RSL, have a three-valued logic where evaluation of a Boolean expression involving chaos leads to everything being undefined!
2.3.2 Sets

Set theory is a branch of mathematical logic that studies sets, which informally are collections of objects. Although any type of object can be collected into a set, set theory is applied most often to objects that are relevant to mathematics. We shall make extensive use of the Zermelo-Fraenkel version (1908, 1921) of set theory. We refer to Appendix Sect. D.1.1 on Page 307 item [7] and Item 7 on Page 308, Sect. D.3.2 on Page 310, and Sect. D.3.6 on Page 312.

2.3.3 Types

**Definition:** By a type [as a noun] we shall mean a possibly infinite set of values of some kind.

We shall take a classical set-theoretic, i.e., Zermelo-Fraenkel, view of types and sort in this monograph. The ‘kind’ is what determines the type. The type of natural numbers, including the number 0, we give the name Nat; the integer type is named Intg; the type of real numbers is named Real. The type of truth values, Booleans, is named Bool.

When defining types, which we shall very often need to do, we shall make use of the RAISE Specification Language (RSL)'s type definition concept. We refer to Appendix Sect. D.1 on Page 307.

We shall often use the term ‘type’ in the specific sense of there being a model for values of the type in the form of either the basic, atomic types given above, or in the form of:

- mathematical sets, $A$-set, $A$-infset,
- Cartesian products, $A \times B \times \cdots \times C$,
- sequences, $A^*$, $A^{\omega}$,
- maps, $A \mapsto B$,
- functions, $A \rightarrow B$ or $A \leadsto \rightarrow B$,

over basic or mathematical values, i.e., types $A, B, C$, etc.

We use the RSL type definition approach. $T$ stands for types. $S$ stands for sorts. $Q$ stands for further undefined atomic values. Recursively defined map and function types are not allowed.

\[
T ::= \text{Bool} \mid \text{Nat} \mid \text{Intg} \mid \text{Real} \mid Q \mid S \\
| T\text{-set} | T\text{-infset} \quad \text{[finite, respectively possibly infinite sets]} \\
| T \times T \times \cdots \times T \quad \text{[Cartesians]} \\
| T^\ast | T^{\omega} \quad \text{[finite, respectively possibly infinite lists]} \\
| T \mapsto T | T \leadsto T \quad \text{[maps, respectively bijective maps]} \\
| T \rightarrow T | T \leadsto T \quad \text{[total, respectively partial functions]}
\]

**Definition:** We shall use the term ‘sort’ to designate a possibly infinite set of values of some further undefined kind.

The term ‘sort’ is commonly used in algebraic semantics [338].

---

4. en.wikipedia.org/wiki/Zermelo-Fraenkel_set_theory
5. Finite sets can be enumerated: \{a_1, a_2, \ldots, a_n\}. Operators are $\in$, $\cup$, $\cap$, $\subseteq$, $\varnothing$, cardinality, etc.
6. Cartesians are expressed as $(a, b, \ldots, c)$.
7. Finite sequences can be enumerated: $(a_1, a_2, \ldots, a_n)$. Operators are $\text{hd}$ (head), $\text{tl}$ (tail), $\text{length}$, $|\cdot|$, $\text{elems}$, etc.
8. Finite maps can be enumerated: $[a_1 \mapsto b_1, a_2 \mapsto b_2, \ldots, a_n \mapsto b_n]$. Operators are: $\cdot(m(a))$, $\text{domain}$, $\text{range}$ (range), etc.
9. Functions are defined: $\lambda x \cdot (\cdot)(x)$, see Example 3 on the following page.
2.3.4 Functions

**Definition:** 18 **Function:** By a function we shall understand ‘something’, which when applied to an argument value of some type, say A, yields a result value of some type, say B (where A and B may be the same type).

**Definition:** 19 **Signature:** By the signature of a function we mean a quadruple: (i) the, or a, name of the function, (ii) either the total function designator, →, or the partial function designator, →, (iii) the type of its argument value(s), and (iv) the type of its result value.

Let the name of (i, iii, iv) be f, A and B, respectively, then we present the signature of the total and the partial functions f as follows:

\[
\text{value } f: A \rightarrow B \quad \text{value } f: A \rightarrow B
\]

**Example 3** A Classical Function Definition A classical function is the factorial function. It can, for example, be defined as follows:

\[
\text{value } f: \text{Nat} \rightarrow \text{Nat}, f(n) \equiv \begin{cases} 1 & \text{if } n = 0 \\ \text{else } \ n \times f(n-1) & \text{end}
\end{cases}
\]

2.3.4.1 Total and Partial Functions

**Definition:** 20 **Total Function:** By a total function we mean a function which is defined for all arguments of its argument type.

**Definition:** 21 **Partial Function:** By a partial function we mean a function which is not defined for all the values of its argument type.

2.3.4.2 Predicate Functions

**Definition:** 22 **Predicate:** By a predicate we mean a function whose result type is Boolean, i.e., `Bool`.

Predicates are, by necessity, total functions.

\[
\cdots
\]

We have listed a number of concepts of mathematical logic. Several of these have related to the possibility of proofs. But a las! In this monograph we shall use the notation of mathematical logic extensively. But not for proofs of properties neither of our descriptions nor of domains. We shall leave the ultimately desirable goal of formulating such properties: invariants and laws, and of their proofs to follow on the heels of this monograph. Before we can run we must learn to walk.

2.3.5 Mathematical Notation versus Formal Specification Languages

2.3.5.1 Mathematics as a Notation – in General

We shall primarily make use of mathematics as a precise notation in which to express ideas about and the prompt calculi — including some, usually not computable, functions. That is: we shall not use mathematics to develop a proper theory of domain analysis & description. For that we refer to [62, Domain Analysis: Endurants – An Analysis & Description Process Model, 2014]. We have indicated issues of axiom systems, and we shall illustrate three axiom systems in this chapter: an Axiom System for Metric Spaces, Sect. 2.4.5 on Page 23; J. van Benthem’s A Continuum Theory of Time, Sect. 2.5.4 on Page 26; and Wayne D. Blizard’s A Theory of Time Space, Sect. 2.5.5 on Page 27.
2.3.5.2 Mathematics as a Notation – in Specific

In many of the more than 100 examples, cf. appendix Sect. E.2 on Page 328 [an index of all examples], those that illustrate formalisation of domains, we use RSL [176], the RAISE Specification Language [179]. But almost elsewhere in the text, in particular in Chapters 3–6, we use a mathematical notation. We comment here on that notation.

Mathematics “as our notation” reflects that the mathematics we rely upon is what is often referred to as discrete mathematics [349, 138, 316, 248, 247, 137, 310, 7]. By discrete mathematics we mean such mathematics, whose disciplines rests set theory and mathematical logic, entails important constructive mathematical structures as

- [i] sets, i.e., definite or indefinite collections of mathematical values, \{a,b,...\}, respectively \{a,b,...\} (where the ..., see below, indicates “and so forth, possibly “ad infinitum”);
- [ii] Cartesians, i.e., definite groupings of mathematical values, \(a,b,...\), of mathematical values;
- [iii] lists, i.e., definite or indefinite sequences of mathematical values, \{a,b,...\}, respectively \{a,b,...\};
- [iv] maps, e.g., \(m\), i.e., explicitly enumerated functions, \([a1\mapsto b1,a2\mapsto b2,...,an\mapsto bn]\), from finite definition sets, \(\text{dom } m = \{a1,a2,...,an\}\), to finite range, or co-domain sets, \(\text{rng } m = \{b1,b2,...,bn\}\), of mathematical values; and
- [v] functions, i.e., \(\lambda\)-definable [125, 102, 15, 16, 17] function values: \(\lambda x.\delta(x)\) where \(x\) is an arbitrary free identifier and \(\delta(x)\) is an arbitrary expression in which \(x\) occurs, usually free – the expressions \(\delta(x)\) otherwise over the kind of values listed above and including respective operators.

The mathematical values include ground values of

- Booleans, false, true, chaos : \textbf{Bool};
- natural numbers, 0, 1, 2, ..., \textbf{Nat};
- integers, ..., -2,-1, 0, 1, 2, ..., \textbf{Intg}; and
- reals, ..., -3.14159265359..., -0.5, ..., 0 ..., 1, +2.71828182846..., ..., \textbf{Real}.

The mathematical values finally include such which are defined through a type definition system “inherited” from RSL. In general our mathematical notation includes many of the clause structures of RSL, structures also found in VDM SL [30, 153, 154], its predecessor, as well as in many programming languages since Algol 60 [257]\(^{10}\) We refer to Appendix Sect. D for details. Example clause structures are:

\[
\begin{align*}
\text{var} & := \text{clause}, \\
\text{if } b & \text{ then } c \text{ else } a \text{ end}, \\
\text{case } p \text{ of } e1 \mapsto c1, ..., \text{ end,} \\
(p1 \mapsto c1, ..., \text{ end),} \\
\text{for } \forall e: E \cdot e \in \text{set do } c(e) \text{ end,} \\
\text{while } e & \text{ do } c \text{ end,} \\
& \text{do } c \text{ while } e \text{ end,} \\
& c1; \text{ ... } cn \text{ and } \\
& \text{skip.}
\end{align*}
\]

That is commensurate with the fact the the formal specification languages, VDM SL [88, 89, 154] and RSL [176], (whose development and initial use, this author has been and is actively involved in since their inception) deeply reflects a discrete mathematics

2.3.5.2.1 The Dot-dot-dot Notation

It is very common, also in strict, “formalistic” mathematics papers to use an inductive form of dot notation: \ldots. Our mathematics notation deploys that good practice. As an interesting paper we refer to [5, Deductive Synthesis of Dot Expressions].

\(^{10}\) Algol W [368], CPL [18], PL/1 [278], Algol 68 [29, 106, 21], Pascal [255, 369], Modula [370, 295], Oberon [371, 372, 373], Ada [31], CHILL [116, 107, 194, 8], Java [184, 342], C# [234], etc.
2.3.5.2.2 The Quote Notation

Here comes an interesting “twist” to our mathematical notation. We refer to the use of **quotes**. By **quoting** an expression, say the expression if b then c else a end, that is, by writing “if b then c else a end” we mean, not the valuation of the unquoted expression, but the text between the quotes; that is, we use the “bracketing” symbols “...” to indicate what, ..., is quoted.

Our need for quoting is motivated as follows: The whole purpose of domain analysis & description is to be able to (i) logically analyse a domain and (ii) produce a textual description of that domain. It is with respect to the description procedures that there is a need for mathematically formally specify that a textual description be yielded. Hence the quotes.

So keep your mind straight when, in Chapters 3–6 we “switch” between mathematical notation’s use of quotes with RSL-like expressions and “clean” RSL with no [need for] quotes.

Historically the use of quoting can be attributed to John McCarthy [279, 280, 281, 1960s][11] and is manifest in Lisp [282]. Lisp’s use of quotes is explained and discussed by Miles Bain in milesmbcain.xyz/the-roots-of--quotation, in en.m.wikipedia.org/wiki/M-expression [Wikipedia], stackoverflow.com/questions/-134887/when-to-use-or-quote-in-lisp and gnu.org/software/emacs/manual/html_node/elisp/Quot- ing.html.

2.3.5.3 An Interplay between Mathematical Notation and Specification Languages

Thus this monograph illustrates a common phenomenon: that research–in–progress, into computing science, sometimes, as here, starts with, as here, domain analysis & description ideas, proceeds with these, making use of mathematical notation, gradually introducing more formal specification language-like notation, while eventually, and thus, as here, implicitly, evolving what looks like a full, formal specification language. We have not found the need, here, to design such a proper domain analyser & describer language. Mathematical notation has no formal syntax and no formal semantics. So, for the time being, “RSL”, and as we shall later introduce, RSL+[12], has no formal syntax and no formal semantics. We leave that to interested readers!

2.4 Space

Mathematicians and physicists model space in, for example, the form of Hausdorf (or topological) space[12]; or a metric space which is a set for which distances between all members of the set are defined; Those distances, taken together, are called a metric on the set; a metric on a space induces topological properties like open and closed sets, which lead to the study of more abstract topological spaces; or Euclidean space, due to Euclid of Alexandria.

2.4.1 Space Motivated Philosophically

**Definition: 23 Indefinite Space:** We motivate the concept of indefinite space as follows: [348, pp 154] “The two relations asymmetric and symmetric, by a transcendental deduction, can be given an interpretation: The relation (spatial) direction is asymmetric; and the relation (spatial) distance is symmetric. Direction and distance can be understood as spatial relations. From these relations are derived the relation in-between. Hence we must conclude that primary entities exist in space. Space is therefore an unavoidable characteristic of any possible world”.

From the direction and distance relations one can derive Euclidean Geometry.

**Definition: 24 Definite Space:** By a **definite space** we shall understand a space with a definite metric.

There is but just one space. It is all around us, from the inner earth to the farthest galaxy. It is not manifest. We can not observe it as we observe a road or a human.

---

Paul Graham: www.paulgraham.com/rootsoflisp.html
Paul Graham: sep.yimg.com/ty/cdn/paulgraham/jmc.ps?t=1564708198&
2.4.2 The Spatial Value

There is an abstract notion of (definite) SPACE(s) of further unanalysable points; and there is a notion of POINT in SPACE.

Type

SPACE

POINT

Space is not an attribute of endurants. Space is just there. So we do not define an observer, \( \text{observe} \_ \text{space} \). For us, bound to model mostly artifactual worlds on this earth there is but one space. Although SPACE, as a type, could be thought of as defining more than one space we shall consider these isomorphic!

2.4.3 Spatial Observers

A point observer, \( \text{observe} \_ \text{POINT} \), is a function which applies to physical endurants, \( e \), and yield a point, \( \ell \). \( \text{POINT} \)

\[ \text{observe} \_ \text{POINT} : E \rightarrow \text{POINT} \]

2.4.4 Spatial Attributes

We suggest, besides POINTs, the following spatial attribute possibilities:

| EXTENT | as a dense set of POINTs; |
| Volume, of concrete type, for example, \( m^3 \), as the “volume” of an EXTENT such that SURFACEs as dense sets of POINTs have no volume, but an Area, of concrete type, for example, \( m^2 \), as the “area” of a dense set of POINTs; LINE as dense set of POINTs with no volume and no area, but Length, of concrete type, for example, \( m \).

For these we have that

| the intersection, \( \cap \), of two EXTENTS is an EXTENT of possibly nil Volume, |
| the intersection, \( \cap \), of two SURFACES may be either a possibly nil SURFACE or a possibly nil LINE, or a combination of these. |
| the intersection, \( \cap \), of two LINES may be either a possibly nil LINE or a POINT. |

Similarly we can define

| the union, \( \cup \), of two not-disjoint EXTENTS, |
| the union, \( \cup \), of two not-disjoint SURFACES, |
| the union, \( \cup \), and of two not-disjoint LINES. |

and:

| the \( \text{in} \)equality, \( \neq \), of pairs of EXTENTS, pairs of SURFACES, and pairs of LINES. |

We invite the reader to first first express the signatures for these operations, then their pre-conditions, and finally, being courageous, appropriate fragments of axiom systems.

2.4.5 Mathematical Models of Space

Figure 2.1 on the next page diagrams some mathematical models of space. We shall hint a just one of these spaces.
**2.4.5.1 Metric Spaces**

A metric space is an ordered pair \((M, d)\) where \(M\) is a set and \(d\) is a metric on \(M\), i.e., a function:

\[
d : M \times M \rightarrow \text{Real}
\]

such that for any \(x, y, z \in M\), the following holds:

1. \(d(x, y) = 0 \iff x = y\) identity of indiscernibles (2.1)
2. \(d(x, y) = d(y, x)\) symmetry (2.2)
3. \(d(x, z) \leq d(x, y) + d(y, z)\) sub-additivity or triangle inequality (2.3)

Given the above three axioms, we also have that \(d(x, y) \geq 0\) for any \(x, y \in M\). This is deduced as follows:

\[
\begin{align*}
d(x, y) + d(y, x) & \geq d(x, x) & \text{triangle inequality (2.4)} \\
d(x, y) + d(y, x) & \geq d(x, x) & \text{by symmetry (2.5)} \\
2d(x, y) & \geq 0 & \text{identity of indiscernibles (2.6)} \\
d(x, y) & \geq 0 & \text{non-negativity (2.7)}
\end{align*}
\]

The function \(d\) is also called distance function or simply distance. Often, \(d\) is omitted and one just writes \(M\) for a metric space if it is clear from the context what metric is used.

**2.5 Time**

a moving image of eternity;  
the number of the movement in respect of the before and the after;  
the life of the soul in movement as it passes from one stage of act or experience to another;  
a present of things past: memory,  
a present of things present: sight,  
and a present of things future: expectations\(^{13}\)

\(^{13}\) Quoted from [12, Cambridge Dictionary of Philosophy]
This thing all things devours:
Birds, beasts, trees, flowers;
Gnaws iron, bites steel,
Grinds hard stones to meal;
Slays king, ruins town,
And beats high mountain down. 14

Concepts of time continue to fascinate philosophers and scientists [353, 152, 284, 318, 319, 320, 321, 322, 323, 324, 335] and [156].

2.5.1 Time Motivated Philosophically

**Definition:** 25 Indefinite Time: We motivate the abstract notion of time as follows. [348, pp 159] “Two different states must necessarily be ascribed different incompatible predicates. But how can we ensure so? Only if states stand in an asymmetric relation to one another. This state relation is also transitive. So that is an indispensable property of any world. By a transcendental deduction we say that *primary entities exist in time. So every possible world must exist in time*”.

**Definition:** 26 Definite Time: By a *definite time* we shall understand an abstract representation of time such as for example year, month, day, hour, minute, second, et cetera.

**Temporal Notions of Endurants**

**Example 4** By temporal notions of endurants we mean time properties of endurants, usually modelled as attributes. Examples are: (i) the time stamped link traffic, cf. Item 173 on Page 99 and (ii) the time stamped hub traffic, cf. Item 169 on Page 98.

2.5.2 Time Values

We shall not be concerned with any representation of time. That is, we leave it to the domain analyser cum describer to choose an own representation [156]. Similarly we shall not be concerned with any representation of time intervals. 15

82 So there is an abstract type `Time`,
83 and an abstract type `TI`: `TimeInterval`.
84 There is no `Time` origin, but there is a “zero” `Time` interval.
85 One can add (subtract) a time interval to (from) a time and obtain a time.
86 One can add and subtract two time intervals and obtain a time interval – with subtraction respecting that the subtrahend is smaller than or equal to the minuend.
87 One can subtract a time from another time obtaining a time interval respecting that the subtrahend is smaller than or equal to the minuend.
88 One can multiply a time interval with a real and obtain a time interval.
89 One can compare two times and two time intervals.

---

14 J.R.R. Tolkien, The Hobbit
15 – but point out, that although a definite time interval may be referred to by number of years, number of days (less than 365), number of hours (less than 24), number of minutes (less than 60), number of seconds (less than 60), et cetera, this is not a time, but a time interval.
2.5.3 Temporal Observers

We define the signature of the meta-physical time observer.

type 90 T

type value 90 record TIME(): Unit → T

The time recorder applies to nothing and yields a time. record,TIME() can only occur in action, event and behavioural descriptions.

Caveat: You may wish to skip the rest of this chapter’s many sections, i.e., Sects. 2.5.4–2.5.10, on time. That may come as a surprise to you. But in our domain modelling we shall refrain from modelling temporal aspects of domains! So why bring all this material? We bring (“all”) this material so that you will know what you are missing! That is, what should also be considered in domain modelling. We therefore leave it to others\textsuperscript{16} to redress this omission. Besides, most of the present work on applying temporal logics in our field has been to software design (requirements).

Modern models of time, by mathematicians and physicists evolve around spacetime\textsuperscript{17} We shall not be concerned with this notion of time.

Models of time related to computing differs from those of mathematicians and physicists in focusing on divergence and convergence, zero (Zenon) time and interleaving time\textsuperscript{377} are relevant in studies of real-time, typically distributed computing systems. We shall also not be concerned with this notion of time.

2.5.4 J. van Benthem

The following is taken from Johan van Benthem [353]: Let $P$ be a point structure (for example, a set). Think of time as a continuum; the following axioms characterise ordering ($<, =, >$) relations between (i.e., aspects of) time points. The axioms listed below are not thought of as an axiom system, that is, as a set of independent axioms all claimed to hold for the time concept, which we are encircling. Instead van Benthem offers the individual axioms as possible “blocks” from which we can then “build” our own time system — one that suits the application at hand, while also fitting our intuition. Time is transitive: If $p < p'$ and $p ' < p''$ then $p < p''$. Time may not loop, that is, is not reflexive: $p \not< p$. Linear time can be defined: Either one time comes before, or is equal to, or comes after another time. Time can be left-linear, i.e., linear “to the left” of a given time. One could designate a time axis as beginning at some time, that is, having no predecessor times. And one can designate a time axis as ending at some time, that is, having no successor times. General, past and future successors (predecessors, respectively successors in daily talk)

\textsuperscript{16} that is: other researchers, lecturers, textbooks

\textsuperscript{17} The concept of Spacetime was first “announced” by Hermann Minkowski, 1907–08 — based on work by Henri Poincaré, 1905–06, \url{https://en.wikisource.org/wiki/Translation:_The_Fundamental_Equations_for_Electromagnetic_Processes_in_Moving_Bodies}
can be defined. Time can be dense: Given any two times one can always find a time between them. Discrete time can be defined.

A Continuum Theory of Time

Axiom System 2

- **TRANS:** Transitivity | ∀ p, p′, p′′ : P ⋀ p < p′ < p′′ ⇒ p < p
- **IRREF:** Irreflexivity | ∀ p : P ⋀ p < p
- **LIN:** Linearity | ∀ p, p′ : P ⋀ (p = p′ ⋀ p < p′ ∨ p > p′)
- **L–LIN:** Left Linearity
  - ∀ p, p′, p′′ : P ⋀ (p < p′ ⋀ p′′ < p) ⇒ (p′ < p′′ ⋀ p′′ = p′ ⋀ p′′ < p′)
- **BEG:** Beginning | ∃ p : P ⋀ ∃ p′ : P ⋀ p < p′
- **END:** Ending | ∃ p : P ⋀ ∼ ∃ p′ : P ⋀ p < p′
- **SUCC:** Successor
  - **PAST:** Predecessors | ∀ p : P ⋀ ∃ p′ : P ⋀ p′ < p
  - **FUTURE:** Successor | ∀ p : P ⋀ ∃ p′ : P ⋀ p < p
- **DENS:** Dense | ∀ p, p′, p′′ : P (p < p′ ⇒ ∃ p′′ : P ⋀ p < p′ < p′′)
- **CDENS:** Converse Dense | ∀ p, p′, p′′ : P (p < p′ ⇒ ∃ p′′ : P ⋀ p < p < p′′)
- **DISC:** Discrete

A strict partial order, SPO, is a point structure satisfying TRANS and IRREF. TRANS, IRREF and SUCC imply infinite models. TRANS and SUCC may have finite, “looping time” models.

2.5.5 Wayne D. Blizard: A Theory of Time–Space

We shall present an axiom system [101, Wayne D. Blizard, 1980] which relate abstracted entities to spatial points and time. Let A, B, . . . stand for entities, p, q, . . . for spatial points, and t, τ for times. 0 designates a first, a begin time. Let t′ stand for the discrete time successor of time t. Let N(p, q) express that p and q are spatial neighbours. Let = be an overloaded equality operator applicable, pairwise to entities, spatial locations and times, respectively. A′ expresses that entity A is at location p at time t. The axioms — where we omit (obvious) typings of A, B, P, Q, and T: t′ designates the time successor function: t′.

A Theory of Time–Space

Axion System 3

- **(I)** ∀A∀t∀p : A′_p
- **(II)** (A′_p ⋀ A′_q) ⊃ p = q
- **(III)** (A′_p ⋀ B′_q) ⊃ A = B
- **(IV)?** (A′_p ⋀ A′_q) ⊃ t = t′
  - **(V i)** ∀ p, q : N(p, q) ⊃ p ≠ q
  - **(V i i)** ∀ p, q : N(p, q) = N(q, p)
  - **(V i ii)** ∀ p′, q, r : N(p, q) ⋀ N(p, r) ⋀ q ≠ r
  - **(VI i)** ∀ t : t ≠ t′
  - **(VI i i)** ∀ t : t′ ≠ 0
  - **(VI i ii)** ∀ t : t ≠ 0 ⋀ ∃ τ : τ = τ′
  - **(VI i iv)** ∀ t, τ : τ′ = t′ ⋀ τ = t
  - **(VII)** A′_p ⋀ A′_q ⊃ N(p, q)
  - **(VIII)** A′_p ⋀ B′_q ⋀ N(p, q) ⊃ (A′_q ⋀ B′_p)

(II–IV,VII–VIII): The axioms are universally ‘closed”; that is: We have omitted the usual ∀A, B, p, q, ts.
(I): For every entity, A, and every time, t, there is a location, p, at which A is located at time t.
(II): An entity cannot be in two locations at the same time.
(III): Two distinct entities cannot be at the same location at the same time.
(IV): Entities always move: An entity cannot be at the same location at different times. This is more like a conjecture: Could be questioned.
(V): These three axioms define N.
(V i): Same as ∀p :∼ N(p, p). “Being a neighbour of”, is the same as “being distinct from”.
(V ii): If p is a neighbour of q, then q is a neighbour of p.
(V iii): Every location has at least two distinct neighbours.
(VI): The next four axioms determine the time successor function \( \cdot \).
(VI i): A time is always distinct from its successor: time cannot rest. There are no time fix points.
(VI ii): Any time successor is distinct from the begin time. Time 0 has no predecessor.
(VI iii): Every non–begin time has an immediate predecessor.
(VI iv): The time successor function \( \cdot \) is a one–to–one (i.e., a bijection) function.
(VII): The continuous path axiom: If entity A is at location p at time t, and it is at location q in the immediate next time (\( t' \)), then p and q are neighbours.
(VIII): No “switching”: If entities A and B occupy neighbouring locations at time t then it is not possible for A and B to have switched locations at the next time (\( t' \)).

Except for Axiom (IV) the system applies both to systems of entities that “sometimes” rests, i.e., do not move. These entities are spatial and occupy at least a point in space. If some entities “occupy more” space volume than others, then we interpret, in a suitable manner, the notion of the point space P (etc.). We do not show so here.

2.5.6 “Soft” and “Hard” Real-time

We loosely identify a spectrum of from “soft” to “hard” temporalities — through some informally worded texts. On that background we can introduce the term ‘real-time’. And hence distinguish between ‘soft’ and ‘hard’ real-time issues. From an example of trying to formalise these in RSL, we then set the course for this chapter.

2.5.7 Soft Temporalities

You have often wished, we assume, that “your salary never goes down, say between your ages of 25 to 65”. How to express that? Taking into account other factors, you may additionally wish that “your salary goes up.” How do we express that? Taking also into account that your job is a seasonal one, we may need to refine the above into “between un-employments your salary does not go down”. How now to express that?

2.5.8 Hard Temporalities

The above quoted (“...”) statements may not have convinced you about the importance of speaking precisely about time, whether narrating or formalising.

So let’s try some other examples:

“The alarm clock must sound exactly at 6 am unless someone has turned it off sometime between 5am and 6 am the same morning.”

“The gas valve must be open for exactly 20 seconds every 60 seconds.”

“The sum total of time periods — during which the gas valve is open and there is no flame consuming the gas — must not exceed one twentieth of the time the gas valve is open.”

“The time between pressing an elevator call button on any floor and the arrival of the cage and the opening of the cage door at that floor must not exceed a given time \( t_{arrival} \).”

The next sections will hint at ways and means of speaking of time.
2.5.9 Soft and Hard Real-time

The informally worded temporalities of “soft real-time” can be said to involve time in a very “soft” way:
No explicit times (e.g., 15:45:00), deadlines (e.g., “27th February 2004”), or time intervals (e.g., “within 2 hours”), were expressed.

The informally worded temporalities of “hard real-time”, in contrast, can be said to involve time in a “hard” way: Explicit times were mentioned.

For pragmatic reasons, we refer to the former examples, the former “invocations” of ‘temporality’, as being representative of soft real-time, whereas we say that the latter invocations are typical of hard real-time.

Please do not confuse the issue of soft versus hard real-time: It is as much hard real-time if we say that something must happen two light years and five seconds from tomorrow at noon!

---

**Soft Real-Time Models Expressed in “Ordinary” RSL Logic**

**Example 5** Let us assume a salary database SDB which at any time records your salary. In the conventional way of modelling time in RSL we assume that SDB maps time into Salary:

```rsl

type
  Time, Sal
SDB = Time ⇒ Sal

value
  hi: (Sal × Sal)((Time × Time) ⇒ Bool)
  eq: (Sal × Sal)((Time × Time) ⇒ Bool)
  lo: (Sal × Sal)((Time × Time) ⇒ Bool)

axiom
  ∀ σ:SDB,t,t':Time • \{t,t'\} ⊆ dom σ ∧ hi(t',t) ⇒ ¬lo(σ(t'),σ(t))
  ∀ t,t':Time •
    (hi(t',t) ≡ (eq(t',t) ∨ lo(t',t))) ∧
    (lo(t',t) ≡ (eq(t',t) ∨ hi(t',t))) ∧
    (eq(t',t) ≡ (lo(t',t) ∨ hi(t',t))) ... /∗ same for Sal /∗
```

---

**Hard Real-Time Models Expressed in “Ordinary” RSL Logic**

**Example 6** To express hard real-time using just RSL we must assume a demon, a process which represents the clock:

```rsl

type
  T = Real

value
  time: Unit ⇒ T
  time() as t

axiom
  time() ≠ time()

The axiom is informal: It states that no two invocations of the time function yields the same value. But this is not enough. We need to express that “immediately consecutive” invocations of the time function yields “adjacent” time points. T provides a linear model of real-time.

variable
  t1,t2 : T
axiom
  □ (t1 := time();
  t2 := time();
  t2 – t1 = /* infinitesimally small time interval: TII*/ ∧
2.5.10 Temporal Logics

“The term Temporal Logic has been broadly used to cover all approaches to the representation of temporal information within a logical framework, and also more narrowly to refer specifically to the modal-logic type of approach introduced around 1960 by Arthur Prior under the name of Tense Logic and subsequently developed further by logicians and computer scientists.”

“Applications of Temporal Logic include its use as a formalism for clarifying philosophical issues about time, as a framework within which to define the semantics of temporal expressions in natural language, as a language for encoding temporal knowledge in artificial intelligence, and as a tool for handling the temporal aspects of the execution of computer programs.”

2.5.10.1 The Issues

The basic issue is simple: To be able to speak of temporal phenomena without having to explicitly mentioning time. That goes for vague, or “soft” notions of time: What we could call “soft real-time”, that something happens at a time, or during a time interval, but with no “fixing” of absolute times nor time intervals. It also, of course, goes for precise, or “hard” notions of time: What we could call “hard real-time”, that something happens at a very definitive point in time, or during a time interval of a very specific length, and thus with “fixing” of absolute times or time intervals.

2.5.10.2 A. N. Prior’s Tense Logics

We present a philosophical linguistics motivated temporal logic. Following the Stanford Encyclopedia of Philosophy, Arthur Prior [322, 323, 324] developed a tense logic along the lines presented below:

- \( Pp \): “It has at some time been the case that \( p \) held”
- \( Fp \): “It will at some time be the case that \( p \) holds”
- \( Hp \): “It has always been the case that \( p \) held”
- \( Gp \): “It will always be the case that \( p \) holds”

\( P \) and \( F \) are known as the weak tense operators, while \( H \) and \( G \) are known as the strong tense operators. The two pairs are generally regarded as inter-definable by way of the equivalences:

\[
Pp \equiv \neg H(\neg p) \\
Fp \equiv \neg G(\neg p)
\]

On the basis of these intended meanings, Prior used the operators to build formulas expressing various philosophical theses about time, which might be taken as axioms of a formal system if so desired. Some examples of such formulas, with Prior’s own glosses (from [323]), are:

\[19\] This and the next slanted quoted text paragraphs are taken from http://plato.stanford.edu/entries/logic-temporal/.

\[20\] http://plato.stanford.edu/entries/prior/
Gp⇒Fp:
What will always be, will be
G(p⇒q)⇒(Gp⇒Gq)
If p will always imply q, then if p will always be the case, so will q
Fp⇒FFp
If it will be the case that p, it will be − in between − that it will be
¬Fp⇒¬Fp
If it will never be that p then it will be that it will never be that p

A special temporal logic is the Minimal Tense Logic Kt. It is generated by the four axioms:

\[ p \Rightarrow H \neg p \]

What is, has always been going to be

\[ p \Rightarrow G \neg p \]

What is, will always have been

\[ H(p \Rightarrow q) \Rightarrow (H \neg p \Rightarrow H \neg q) \]

Whatever always follows from what always has been, always has been

\[ G(p \Rightarrow q) \Rightarrow (G \neg p \Rightarrow G \neg q) \]

Whatever always follows from what always will be, always will be

### 2.5.10.3 The Duration Calculus

The duration calculus, DC, is due to Zhou Chao Chen, C.A.R. Hoare, Anders P. Ravn, Michael Reichhardt Hansen and others. The definitive introductory work on DC is [380]. We present a terse summary.

#### 2.5.10.3.1 A Function & Safety Example

We show a classical example.

1. For a lift system to be adequate it must always be safe and function adequately. There are three functional requirements.

2. For the lift system to be safe, then for any duration that the door on floor i is open, the lift must be also at that floor.

3. The length of time between when someone pushes a button, inside a lift cage, to send it to floor i, and the arrival of that cage at floor i must be less than some time \( t_s \).

4. The length of time between when someone pushes a button, at floor i, to call it to that floor, and the arrival of a cage at that floor must be less than some time \( t_c \).

5. The length of time that a door is open when a cage is at floor i must be at least some time \( t_o \).

\[ \text{Req} \equiv \Box(\text{SafetyReq} \land \text{FunctReq1} \land \text{FunctReq2} \land \text{FunctReq3}) \]

\[ \text{SafetyReq} \equiv [\text{door}=i] \Rightarrow [\text{floor}=i] \]

\[ \text{FunctReq1} \equiv \left( [i \in \text{send}] ; \text{true} \Rightarrow (\ell \leq t_s) \lor (\ell \leq t_c) \lor [\text{door}=i] ; \text{true} \right) \]

\[ \text{FunctReq2} \equiv \left( [i \in \text{call}] ; \text{true} \Rightarrow (\ell \leq t_s) \lor (\ell \leq t_c) \lor [\text{door}=i] ; \text{true} \right) \]

\[ \text{FunctReq3} \equiv [\text{door}\neq i] ; [\text{door}=i] ; [\text{door}\neq i] \Rightarrow \ell \geq t_o \]

#### 2.5.10.3.2 The Syntax

We only present an overview of the DC syntax. The presentation of this part follows that of Skakkebæk et al. [343] (1992).
2.5.10.3.2.1 **Simple Expressions:** We define simple, i.e., atomic expressions.

\[
x, y, \ldots, z: \text{State Variable}
\]

\[
a, b, \ldots, c: \text{Static Variable}
\]

\[
ff, tt: \text{Bool Const}
\]

\[
k, k', \ldots, k'': \text{Const}
\]

Static variables designate time-independent values. We assume some context which helps us determine the type of variables.

2.5.10.3.2.2 **State Expressions and Assertions:** We define state expressions and state assertions. A state assertion is a state expression of type \text{Bool}, and \text{op} is an operator symbol of arity \( n \). We assume a context which helps us determine that an identifier is an \text{op}?

\[
\text{se: State Expr ::= Const | Bool Const | op(se_1, \ldots, se_n)}
\]

\[
\text{P: State Asrt ::= State Expr}
\]

We assume a context which helps us determine that a state expression is of type \text{Bool}, i.e., is a state assertion.

2.5.10.3.2.3 **Durations and Duration Terms:** If \( P \) is a state assertion, then \( \int P \) is a duration.

We define duration terms.

\[
\text{dt: Dur Term ::= \int P | \text{Real} | op(dt_1, \ldots, dt_n) | \ell}
\]

\( \ell \) is an abbreviation for the duration term \( \int tt \). \text{op} is an \( n \) operator symbol of type \text{Real}. We assume a context which helps us determine that an identifier is an \text{op}.

2.5.10.3.2.4 **Duration Formulas:** We define duration formulas. Let \( A \) be any \( n \)-ary predicate symbol over real-valued duration arguments. We assume a context which helps us determine that an identifier is an \( A \).

\[
\text{d: Dur Form ::= A(dt_1, \ldots, dt_n)}
\]

\[
| \text{true} | \text{false} |
\]

\[
| \neg d | d_1 \lor d_n |
\]

\[
| d_1 \land d_n |
\]

\[
| d_1 \Rightarrow d_n |
\]

\[
| d_1 \land d_n |
\]

\[
| \forall a: d /\ast a is * / \text{Static Variable} |
\]

Delimiting parentheses can be inserted to clarify precedence.

2.5.10.3.2.5 **Common Duration Formula Abbreviations:** We make free use of the following common abbreviations:

\[
| [] : \ell = 0 : \text{point duration} |
\]

\[
| [P] : \int P = \ell \land \ell > 0 : \text{almost everywhere } P |
\]

\[
| \diamond d : \text{true}; d; \text{true} : \text{somewhere } d |
\]

\[
| \neg (\diamond \neg d) : \text{always } d |
\]

\[
| d_1 \rightarrow d_2 : d_1; \text{true} \Rightarrow d_1 \lor (d_1; 1; d_2; \text{true}) : d_2 \text{ follows } d_1 |
\]

**Precedence Rules:**

First : \( \neg \square \diamond \)

Second : \( \lor \land ; \)

Third : \( \Rightarrow \rightarrow \)
2.5.10.3.3 Discussion: From Domains to Designs

We have covered core aspects of the Duration Calculus. The Duration Calculus offers a logic based on intervals and real-time. One can use the Duration Calculus to abstractly express constraints, i.e., requirements, on the duration of states. One can also use the Duration Calculus to abstractly express properties of the domain, i.e., of the application area for which software is sought. And one can finally hint at major design decisions also using the Duration Calculus.

Only in a very implicit sense can Duration Calculus expressions be said to specify sequential programs — such as we are normally prepared to implement in computing systems: in terms of sequential programs. A Duration Calculus expression, however, usually implies a sequential program, or a set of cooperating such. RSL specifications, the “closer” we get to software design, i.e., the more “concrete” such specifications become, rather specifically specify sequential programs. At least, it would be a good idea for the developer to make sure that this is so!

Now how can we combine the ability of the Duration Calculus to express quantitative properties of software (to be designed) and the actual specification of such software?

We turn to this question next. That is, we may seem to completely abandon thoughts and concepts of Duration Calculus, in favour of rather “down to earth” concepts of explicit timing in what could be considered a specification programming language, Timed RSL, TRSL.

2.6 Spatial and Temporal Modelling

It is not always that we are compelled to endow our domain descriptions with those of spatial and/or temporal properties. In our experimental domain descriptions, for example, [67, 95, 71, 69, 42, 55, 64, 36], we have either found no need to model space and/or time, or we model them explicitly, using slightly different types and observers than presented above. We have brought this material on various temporal logics in order to strongly hint at their being used in domain modelling — so there is an interesting challenge!

2.7 Matter

Space, in the sense of SPACE, is “inhabited”! The inhabitants are the entities that Kai Sørlander’s Philosophy refers to, Page 13. They possess properties about which we reason. We shall take the view that these entities are of MATTER. Matter is anything that has mass and takes up space.

The modelling of matters, sometimes referred to as MATTER, is done primarily by means of attributes. We refer to a future, extensive section, Sect. 4.4, on Attributes. But already here is a good place to discuss the ‘matter’ of matter! How does matter manifest itself to you, a human mortal, the domain analyser & describer? You, yourself, your body, is a manifestation of matter. The room, you are in, is matter. The things in it, each are matter. The outdoor environment, in which you walk, is matter. Is the air, you breathe, matter? Yes we say! Is the atmosphere matter? Yes indeed. Really? Does atmosphere have mass? Yes, indeed!

There is a notion: substance theory. We shall not discuss its possible rôle here. But we shall take the liberty of sometimes using the term ‘substance’ in lieu of the term ‘matter’.

2.8 Identity and Mereology

Identity, as a philosophical issue, has emerged from Kai Sørlander’s Philosophy, Chapter 1. We shall make capital use of that concept in this monograph. Mereology, as a philosophical and logic issue, was studied

21 the troposphere, stratosphere, ozone layer, mesosphere, thermosphere, ionosphere, exosphere, ...
22 Substance theory, or substance-attribute theory, is an ontological theory about object-hood positing that a substance is distinct from its properties. A thing-in-itself is a property-bearer that must be distinguished from the properties it bears [Wikipedia].
2.8.1 Identity

It is a fact, that is, an absolutely necessary condition for our description of any world that its entities have unique identity. It is, however a problem in our domain analyser & describer, to secure such identity; so we must, wherever necessary present axioms expressing so. This will be done in Sect. 4.2. A further treatment of mereology is given in Appendix B.

2.8.2 Mereology: Philosophy and Logic

“Mereology (from the Greek µερος ‘part’) is a theory of part-hood relations: of the relations of part to whole and the relations of part to part within a whole”\(^{23}\).

2.8.2.1 Mereology Understood Spatially

In this contribution we restrict ‘parts’ to be those that, firstly, are spatially distinguishable, then, secondly, while “being based” on such spatially distinguishable parts, are conceptually related. We use the term ‘part’ in a more general sense than in [70]. The relation: “being based”, shall be made clear in this paper. Accordingly two parts, \(p_x\) and \(p_y\), (of a same “whole”) are are either “adjacent”, or are “embedded within”, one within the other, as loosely indicated in Fig. 2.2. ‘Adjacent’ parts are direct parts of a same third part, \(p_z\), i.e., \(p_x\) and \(p_y\) are “embedded within” \(p_z\); or one \((p_x)\) or the other \((p_y)\) or both \((p_x\) and \(p_y\)) are parts of a same third part, \(p_z^2\) “embedded within” \(p_z\); et cetera; as loosely indicated in Fig. 2.2, or one is “embedded within” the other — etc. as loosely indicated in Fig. 2.3.

Parts, whether ‘adjacent’ or ‘embedded within’, can share properties. For adjacent parts this sharing seems, in the literature, to be diagrammatically expressed by letting the part rectangles “intersect”. Usually properties are not spatial hence ‘intersection’ seems confusing. We refer to Fig. 2.4.

Fig. 2.4. Two models, [L,R], of parts sharing properties

Instead of depicting parts sharing properties as in Fig. 2.4[L]eft, where shaded, dashed rounded-edge rectangles stands for ‘sharing’, we shall (eventually) show parts sharing properties as in Fig. 2.4[R]ight where –– connections connect those parts.

2.8.2.2 Our Informal Understanding of Mereology

Mereology, to us, is the study and knowledge about how physical and conceptual parts relate and what it means for a part to be related to another part: being disjoint, being adjacent, being neighbours, being contained properly within, being properly overlapped with, et cetera.

By physical parts we mean such spatial individuals which can be pointed to. Examples: a road net (consisting of street segments and street intersections); a street segment (between two intersections); a street intersection; a road (of sequentially neighbouring street segments of the same name); a vehicle; and a platoon (of sequentially neighbouring vehicles).

By a conceptual part we mean an abstraction with no physical extent, which is either present or not. Examples: a bus timetable (not as a piece or booklet of paper, or as an electronic device, but) as an image in the minds of potential bus passengers; and routes of a pipeline, that is, neighbouring sequences of pipes, valves, pumps, forks and joins, for example referred to in discourse: “the gas flows through “such-and-such” a route”. The tricky thing here is that a route may be thought of as being both a concept or being a physical part — in which case one ought give them different names: a planned route and an actual road, for example.

The mereological notion of subpart, that is: contained within can be illustrated by examples: the intersections and street segments are subparts of the road net; vehicles are subparts of a platoon; and pipes, valves, pumps, forks and joins are subparts of pipelines.

The mereological notion of adjacency can be illustrated by examples. We consider the various controls of an air traffic system, cf. Fig. B.1 on Page 289, as well as its aircraft, as adjacent within the air traffic system; the pipes, valves, forks, joins and pumps of a pipeline, cf. Fig. B.6 on Page 292, as adjacent within the pipeline system; two or more banks of a banking system, cf. Fig. B.3 on Page 291, as being adjacent.

The mereo-topological notion of neighbouring can be illustrated by examples: Some adjacent pipes of a pipeline are neighbouring (connected) to other pipes or valves or pumps or forks or joins, et cetera; two immediately adjacent vehicles of a platoon are neighbouring. The mereological notion of proper overlap can be illustrated by examples some of which are of a general kind: two routes of a pipelines may overlap; and two conceptual bus timetables may overlap with some, but not all bus line entries being the same; and some really reflect adjacency: two adjacent pipe overlap in their connection, a wall between two rooms overlap each of these rooms — that is, the rooms overlap each other “in the wall”.

2.9 A Foundation

This, then, is the foundation upon which this monograph is built: the Concepts, as outlined in Chapter 0, the Philosophy of Kai Sørlander, as outlined in Chapter 1, and Logic and Mathematics, the closer
inspection of the concepts **SPACE, TIME and MATTER**, and **Identity** and **Mereology** of the present chapter.
Chapter 3–7 Overview

In the next four chapters “we introduce a domain science & engineering” of domain analysis & description. These four chapters treat “more-or-less” separable core topics of domain analysis & description. The treatment focuses on what we shall call the intrinsics, the essentials of domains.

- Chapter 3 introduces the concepts of entities, endurants and perdurants and unveils basic principles and techniques for the analysis & description of what we shall call external endurant qualities, those which we can experience by looking at them!
- Chapter 4 unveils basic principles and techniques for the analysis & description of what we shall call internal endurant qualities, like identity, mereology or those, attributes, which we can otherwise measure by physical instruments or which record events.
- Chapter 5 provides a bridge between the principles and techniques of Chapters 3–4 and Chapter 6 by further elaborating on the idea of transcendental deduction first introduced in Chapter 1; while also covering the concepts of space and time, not as metaphysical phenomena, but, with our background in Kai Sørlander’s Philosophy, as rational, transcendentally understood concepts.
- Chapter 6, finally, concludes the basic domain analysis & description principles and techniques by transcendentally relating endurants, the “still” entities, observable in space, to perdurants, the “discrete dynamic” entities, observable also in time: actions, events and behaviours.

Chapters 3–4 show you how to systematically develop descriptions of the structure and values of domains while Chapter 6 shows you how to follow that up with the development of core aspects of the behaviour of domains.

- Chapter 7 covers some principles and techniques of mostly non-intrinsical domain entities.
A Theory, not The Theory

We write: “we introduce a domain science & engineering ...”. By this we mean: “there may be other such domain science & engineering”! The science & engineering we refer to are the domain analysis & description prompts. We shall introduce these analysis & description prompts in the next four chapters. But there could be another composition than the one we offer. The present one has served well in guiding the domain analysis & description of many domains [80]. The notions of conjoins, for example, are based very much on a mixture of observations and pragmatism, and, as such could be replaced! Anyway, the reader is guided, in the next 4 chapters, into this ontology and should, from there, be able to modify that ontology to suit the problem at hand.

On Learning a Theory and On Learning a Method

In this monograph we aim to develop a theory of domain analysis & description, and to develop a method of analysing & describing domains. The two things are not quite the same, but, obviously, related. It is important that the reader understands the next subsections. Because of the double aim it is possible that the reader misses the distinction, and hence to learn either!

First you learn about it! Then you learn to do it!

Towards a Theory of Domain Analysis & Description

Before one can practice a method one must learn (i) its possible theoretical basis.

In this monograph the text from which you should learn about a theoretical basis for domain analysis & description is interwoven with the text from which you should learn about the method, i.e. the practice of applying some of the theory. Being interwoven may mean that the reader forgets what it is that is being communicated.

The unfolding of the “story” of the possible theoretical basis is careful, “slow”, almost pedantic. It is perhaps easy to get lost and forget that other thing to be learned, (ii), the method.

A Method for Domain Analysis & Description

The method, to repeat, embodies principles, techniques and tools for the analysis & description of domains.

So how does one proceed in doing a domain model? First one determines what the domain is, i.e., an endurant. Then you analyse it as prescribed. At one point you are then ready to describe the root of the domain, as a composite, or as a structure, or as a material. That description leads to the analysis of sub-endurants, usually several. In a single person project you therefore have to put some of these ‘several’ other endurants on hold, say put as a reminder on what we shall call a notice board. Then later, again an again, from the subsequent analysis & description of these other endurants emerges yet more endurants to be analysed & described. Etcetera.

The method involves an iterative process.

For the professional, practicing domain analyser & describer when, as is covered in Chapter 3, analyses & describes the external qualities of endurants that person is fully aware of their being also internal qualities to analyse & described, and also, subsequently, their transcendental deduction into perdurants: channels, variables and behaviours. In Chapter 3, however, we pose exercise problems, at a stage where the problem solver has yet to learn, as in subsequent chapters, 4–6, about internal qualities, such which ultimately decide on the sort of an endurant; thus the problem solver is, to an extent, disadvantaged; hence may, after e.g. Chapter 4 (etc.), have to return to “improve” on a proposed problem solution. It cannot be otherwise.

These are some analysis prompts

- entity pg. 43, is_entity pg. 43
- endurant pg. 47, is_endurant pg. 47
- perdurant pg. 48, is_perdurant pg. 48
- discrete pg. 48, is_discrete pg. 48
- continuous pg. 48, is_continuous pg. 48
- physical part pg. 50, is_physical part pg. 50
- structure pg. 50, is_structure pg. 50
- living species pg. 51, is_living species pg. 51
- natural part pg. 51, is_natural part pg. 51
- artefact pg. 51, is_artefact pg. 51
- plant pg. 53, is_plant pg. 53
- animal pg. 54, is_animal pg. 54
- human pg. 54, is_human pg. 54
- atomic pg. 56, is_atomic pg. 56
- composite pg. 56, is_composite pg. 56
- conjoin pg. 57, is_conjoin pg. 57
- part material pg. 57, is_part material pg. 57
- material parts pg. 58, is_material parts pg. 58
- part parts pg. 59, is_part parts pg. 59
3

DOMAINS – A Taxonomy

External Qualities

In this chapter we introduce the concepts of endurants and perdurants, the concept of external qualities\(^1\) of endurants, and cover the analysis and description of external qualities of endurants.

Our focus is on domains. So what are domains?.

3.1 Overview of this Chapter

• This is a large chapter.
  • It spans Pages 39–83.
  • To help the reader we present this overview.
  • Some sections can be “armchair-read”.
  • They introduce overall concepts.
  • These are Sects. 3.2, 3.3, 3.5, 3.14 and 3.16.
  • Section 3.21 presents background theory material.
  • It can be skipped, but, when read, must be read carefully.
  • Sections 3.6–3.15 and 3.17 form a first “half” of serious study sections on the ontology of entities and the analysis of external qualities of endurants.
  • Sections 3.18–3.20 form the second “half” of serious study sections on the description of external qualities of endurants.

3.2 Domains

**Definition: 27** Domain: By a domain we shall understand a rationally describable segment of a discrete dynamics segment of a human assisted reality, i.e., of the world, its physical parts: natural [“God-given”] and artefactual [“man-made”], and its living species: plants and animals including, notably, humans. These are endurants (“still”), as well as perdurants (“alive”). Emphasis is placed on “human-assisted”, that is, that there is at least one (man-made) artefact and, therefore, that humans are a primary cause for change of endurant states as well as perdurant behaviours.

This is a terse, but not a fully satisfactory characterisation. But it is the best we can come up with! Let us examine it in some detail.

• By a rational description we mean: a description which is logical, that is, a description over which one can reason; furthermore we shall in addition to this, by rational, mean a description which otherwise deploys additional mathematical concepts.

• By discrete dynamics we mean: a behaviour of the domain which, over time, varies, but in discrete steps: endurant entities may move or change form in space, values of endurant mereologies may vary, and/or values of endurant attributes may vary.

\(^1\) We refer to Definition 29 on Page 42 [Sect. 3.4] for an attempt to define the concept of external quality.
Control theory, the study of the control of continuously operating dynamical systems in engineered processes and machines, is one thing; domain engineering is “a different thing”. Control theory builds upon classical physics, and uses classical mathematics, partial differential equations, etc., to model phenomena of physics and therefrom engineered ‘machines’. Domain science & engineering, in some contrast, builds upon mathematical logic, and, to some extent, modern algebra, to model phenomena of mostly artefactual systems.

- By “a reality” we mean: that which we, as humans, with our senses, can see, hear, smell, taste and touch — as well as that for which we humans have devised apparatuses that measure: mass (kg), time interval (s), temperature (K), electric current (A), amount of substance (mol), luminous intensity (cd), and distance (m).
- By “a human assisted reality” we mean: a world in which focus is on man made endurants and human instigated actions, events and behaviours.

The other technical terms will be explained more formally in the rest of this chapter.

**Definition:**

**Domain Description:** By a domain description we shall understand a combination of narration and formalisation of a domain. A formal specification is a collection of sort, or type definitions, function and behaviour definitions, together with axioms and proof obligations constraining the definitions. A specification narrative is a natural language text which in terse statements introduces the names of (in this case, the domain), and, in cases, also the definitions, of sorts (types), functions, behaviours and axioms; not anthropomorphically, but by emphasizing their properties.

Domain descriptions are (to be) void of any reference to future, contemplated software, let alone IT systems, that may then support entities of the domain. As such domain models can be studied separately, for their own sake, for example as a basis for investigating possible domain theories, or can, subsequently, form the basis for requirements engineering with a view towards development of (‘future’) software, etc. Our aim is to provide a method for the precise analysis and the formal description of domains.

### 3.3 Universe of Discourse

By a universe of discourse we shall understand the same as the domain of interest, that is, the domain to be analysed & described.

#### Universes of Discourse

**Example 7** We refer to a number of Internet accessible experimental reports of descriptions of the following domains:

- railways [35, 85, 38],
- container shipping [42],
- stock exchange [55],
- oil pipelines [60],
- “The Market” [36],
- Web systems [54],
- weather information [67],
- credit card systems [64],
- document systems [71],
- urban planning [95],
- swarms of drones [69],
- container terminals [73]

\[\text{— but it may be that a domain being analysed} \quad \text{— described depends crucially on IT and software} \quad \text{— in which case that must somehow, “ever so abstractly”, be described} !\]

\[\text{We use the terms ‘domain descriptions’ and ‘domain models’ interchangeably.}\]
Method Step 1 Select Domain of Interest:

A principle of the method is, as an initial step of the development of a domain analyser & describer, is to select the universe of discourse, to ascribe it a sort name, say UoD, and to remember that that universe, and, as a technique, be subject to analysis & description.

<table>
<thead>
<tr>
<th>Domain Description Prompt 1</th>
<th>name_and_sketch_universe_of_discourse:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naming:</td>
<td>name_and_sketch_universe_of_discourse</td>
</tr>
<tr>
<td>type UoD</td>
<td></td>
</tr>
<tr>
<td>Rough Sketch:</td>
<td>informal text ...</td>
</tr>
</tbody>
</table>

A Road Transport Domain

Example 8

Naming: type RTS

Rough Sketch: The road transport system that we have in mind consists of a road net and a set of vehicles such that the road net serves to convey vehicles. We consider the road net to consist of hubs, i.e., street intersections, or just street segment connection points, and links, i.e., street segments between adjacent hubs. We consider vehicles to consist of departments of motor vehicles, bus companies, each with zero, one or more buses, and vehicle associations, each with zero, one or more members who are owners of zero, one or more vehicles.

It may be a “large” domain, that is, consist of many, as we shall see, endurants and perdurants, of many parts and materials, of many humans and artefacts, and of many actors, actions, events and behaviours.

Or it may be a “small” domain, that is, consist of a few such entities.

The choice of “boundaries”, that is, of how much or little to include, and of how much or little to exclude is entirely the choice of the domain engineer cum scientist: the choice is crucial, and is not always obvious.

The choice delineates an interface, that is, that which is within the boundary, i.e., is in the domain, and that which is without, i.e., outside the domain, i.e., is the context of the domain, that is, the external domain interfaces. Experience helps set reasonable boundaries.

There are two “situations”: Either a domain analysis & description endeavour is pursued in order to prepare for a subsequent development of requirements modelling, in which case one tends to choose a “narrow” domain, that is, one that “fits”, includes, but not much more, the domain of interest for the requirements. Or a domain analysis & description endeavour is pursued in order to research a domain. Either one that can form the basis for subsequent engineering studies aimed, eventually at requirements development; in this case “wider” boundaries may be sought. Or one that experimentally “throws a larger

---

4 In this monograph we bring several categories of numbered examples. There are the examples, let us call them the text explanatory examples; then there are two kinds of examples related to domains: the informal domain examples and formal domain examples. The latter exemplify the kind of domain analysis & descriptions this monograph studies and for which this monograph presents a method for their construction. We leave it to the reader to discern which examples are what.

5 These are draft reports, more-or-less complete. The writing of these reports was finished when sufficient evidence, conforming or refuting one or another aspect of the domain analysis & description method.
3.4 External Qualities

Definition: 29 External Quality: By an external quality of an endurant we shall, initially, mean a property that is manifest, one that can be touched or seen, generally, one that “forms” the endurable entities of a domain.

More generally, by an external quality of an endurant we shall mean an abstract property about a collection of manifest entities, like a structure of manifest entities, or a structure of abstracted such entities.

A Road Transport System: Manifest External Qualities

Example 9 Our intention is that the manifest external qualities of a road transport system are those of its roads, their hubs i.e., road (or street) intersections, and their links, i.e., the roads (streets) between hubs, and vehicles, i.e., automobiles – that ply the roads – the buses, trucks, private cars, bicycles, etc.

A Road Transport System: Abstract External Qualities

Example 10 Examples of what could be considered abstract external qualities of a road transport domain are: the aggregate of all hubs and all links, the aggregate of all buses, say into bus companies, the aggregate of all bus companies into public transport, and the aggregate of all vehicles into a department of vehicles. Some of these aggregates may, at first be treated as abstract. Subsequently, in our further analysis & description we may decide to consider some of them as concretely manifested in, for example, actual departments of roads.

Method Step 2 External Qualities:

6 We have highlighted certain endurant sort names – as they will re-appear in rather many upcoming examples.
An important step in the process of unfolding an analysis & description of a domain is to determine which are the external qualities of entities of that domain. Our attempt, in Definition 29 on the facing page, to encircle the ‘external quality’ concept may not be fully satisfactory. We shall try “repair” that “failure to be precise” by numerous examples – and otherwise hope that some readers can suggest improved definitions.

We refer to Fig. 3.1 on the next page where a largest dashed-line “upper left” box indicate, in a way, the concepts entailed by external qualities.

### 3.5 Entities

A core concept of domain modelling is that of an entity.

#### Definition: 30 Entity

By an entity we shall understand a phenomenon, i.e., something that can be observed, i.e., be seen or touched by humans, or that can be conceived as an abstraction of an entity; alternatively, a phenomenon is an entity, if it exists, it is “being”, it is that which makes a “thing” what it is: essence, essential nature [268, Vol. I, pg. 665].

#### Analysis Predicate Prompt 1 is_entity:

The domain analyser analyses “things” (θ) into either entities or non-entities. The method provides the domain analysis prompt:

- **is_entity** – where is_entity(θ) holds if θ is an entity.

is_entity is said to be a prerequisite prompt for all other prompts. Prompts, whether analysis or description prompts, are aidé-memoires; they are not program constructs, they can not be defined mathematically; think of them as written on the wall of your working place, there to remind you of what you should remember to do.

#### Method Step 3 “What can be Described”:

A next step in the development of a domain analyser & describer is to decide on what can be described. Both with respect to the universe of discourse and with respect to every subsequently identified entity. The is_entity analysis prompt is the tool used to prompt that analysis and decision. The domain analysers has great leeway here. They can, perhaps rather arbitrarily, some would say, magisterially, decide on leaving out phenomena for further treatment, phenomena that others would say can be described. An excuse for exclusion could be that the domain analysers can claim that the phenomenon is not relevant to their inquiry.

To sum up: An entity is what we can analyse and describe using the analysis & description prompts outlined in this chapter. Other words for ‘entity’ are: ‘material object’ or ‘thing’. Since we shall be needing the term ‘material’ for a specific class of entities, and since the term ‘object’ is already heavily overloaded, we shall just use the term ‘entity’.

The entities that we are concerned with are those with which Kai Sørlander’s Philosophy is likewise concerned. They are the ones that are unavoidable in any description of any possible world. And then, which are those entities? In both [345] and [348] Kai Sørlander rationally deduces that these entities must be in space and time, must satisfy laws of physics – like those of Newton and Einstein, but among them are also living species: plants and animals and hence humans. The living species, besides still being in space and time, and satisfying laws of physics, must satisfy further properties.

---

7 marks the end of a analysis prompt definition.

8 See however [62, 65] where we do suggest an underlying mathematical model of domains and give prompts a semantics.
Figure 3.1 shows an upper ontology for domains such as we shall focus on in this monograph. We shall briefly review Fig. 3.1 by means of a top-down, left-traversal of the tree (whose root is at the top).

3.5.1 A Linnean, Binomial Taxonomy

We shall first present an idealised form of ontology for the domains that we are interested in studying and for whose construction of domain analyses & descriptions we wish to present a method. This idealised form follows that of Carl von Linné (Carl Linnaeus, 1707–1778) the Swedish botanist, zoologist, and physician who formalised binomial nomenclature, the modern system of naming organisms. He is known as the “father of modern taxonomy”.

We refer to the ontology description that now follows, as ‘ideal’. It is so, we claim, because it is strictly binomial, and it is, in a sense, and also as a result of being binomial, abstract in that it does not reflect how it should preferably be used. We shall later, on the basis of the following taxonomy, present a workable, we claim, practically useful ontology.

An ‘Idealised’ Domain Taxonomy

[0] Universes of discourse consists of non-describable phenomena [1.0] and describable phenomena [1.1]. [1.0] Non-describable phenomena will here be left further un-analysed. [1.1] Describable phenomena, are also called entities [1.2].

9 An upper ontology (in the sense used in information science) consists of very general terms that are common across all domains. [Wikipedia].

10 We could organise the ontology differently: entities are either naturals, artefacts or living species, etcetera. If an upper node (•) satisfies a predicate $\exists$ then all descendant nodes do likewise.
A Cursory Overview

A Foundation for Software Development

[1.2] Entities are either **endurants**\(^{11}\) [2.1] or **perdurants**\(^{12}\) [2.2].

[2.1] Endurants are either **physical**\(^{13}\) [3.1] or **living species**\(^{14}\) [3.2].

[3.1] Physical endurants are either **singular**\(^{15}\) [4.1] or **composites**\(^{16}\) [4.2].

[4.1] Singulars are either **atomic parts**\(^{17}\) [5.1] or **materials**\(^{18}\) [5.2].

[5.1] Atomic parts are presently left further un-analysed.

[5.2] Materials are presently left further un-analysed.

[4.2] Composites are either **definite composites**\(^{19}\) or **indefinite composites**\(^{20}\).

[3.2] Living species are either **plants**\(^{21}\) [4.3] or **animals**\(^{22}\) [4.4].

[4.3] Plants are here left further un-analysed.

[4.4] Animals are either **humans** [5.3] or **other** ... [5.4].

[5.3] Humans are here left further un-analysed.

[5.4] Other ... is here left further un-analysed.

[2.2] Perdurants are either **instantaneous**\(^{23}\) [3.3] or **prolonged**\(^{24}\) [3.4].

[3.3] Instantaneous perdurants are either **actions**\(^{25}\) [4.5] or **events**\(^{26}\) [4.6].

[4.5] We shall here leave actions further un-analysed.

[4.6] We shall here leave events further un-analysed.

[3.4] We shall here rename prolonged perdurants into **behaviours**\(^{27}\)

[3.4] Behaviours are here left further un-analysed.

The above textual listing is rendered graphically in Fig. 3.2 on the next page.

Figure 3.5.1 on the facing page shows the basic relational structure of general domain concepts. Figure 3.1 on the preceding page, in principle, builds on the taxonomy of Fig. 3.5.1 on the facing page. The ontology of Fig. 3.1 is “massaged” with respect to Fig. 3.5.1 on the preceding page. Some domain analysis & description concepts have been added; some “intermediary” concepts have been inserted, and, most importantly, the ‘taxonomy’ has evolved into an ‘ontology’. Where the taxonomy only dealt with tangible, visible properties, the ontology ‘adds’ intangible, but objectively measurable properties shown by the bottom vertical **unique identifier**, **mereology** and **attribute lines**. We shall now proceed to justify Fig. 3.1.

**3.5.2 A Cursory Overview**

There are **describable** phenomena and there are phenomena that we cannot describe. The former we shall call **entities**. The **entities** are either **endurants**, “still” entities – existing in **space**, or are **perdurants**, “alive” entities – existing also in **time**. **Endurants** are either **discrete** or **continuous** – in which latter case we call them **materials**. **Discrete endurants** are **physical parts**, or **living species**, or are **structures**. **Physical parts** are either **naturals**, or **artefacts**, i.e. man-made. Natural parts are either **atomic** or **composite parts**. Man-made parts are either **atomic parts**, **composite parts** or are **conjoins**. In this monograph we shall refer to man-made parts as **artefacts**. **Conjoins** are either **part-materials**, or **material-parts**, or **part-parts** conjoins. **Living**

---

\(^{11}\) Cf. Defn. 31 on Page 47

\(^{12}\) Cf. Defn. 32 on Page 47

\(^{13}\) Physical endurants are such entities which can alone be justified in terms of physical laws such as Newton’s etc.

\(^{14}\) Living species are such entities which, in addition to physical laws are also subject to biological laws.

\(^{15}\) By a singular entity we shall mean a single instance or something to be considered by itself [Merriam Webster].

\(^{16}\) By a constellation we shall mean a grouping of usually two or more endurants.

\(^{17}\) See Sect. 3.13.2 on Page 56

\(^{18}\) Cf. Defn. 48 on Page 54

\(^{19}\) A definite composite has a given number of endurants.

\(^{20}\) An indefinite composite has a possibly varying number of endurants.

\(^{21}\) See Sect. 3.11.1 on Page 53

\(^{22}\) See Sect. 3.11.2 on Page 54

\(^{23}\) An Instantaneous Perdurant occurs at a (or any) single point in time and manifests itself in a similarly instantaneous state change – where a state is the internal qualities value of any assembly of endurants.

\(^{24}\) A Prolonged Perdurant occurs over time, perdures for either an indefinite or an infinite time interval.

\(^{25}\) An action is an internally provoked instantaneous state change.

\(^{26}\) An event is an externally provoked instantaneous state change.

\(^{27}\) A behaviour is a set of sequences of actions, events and behaviours.

\(^{28}\) – like those used as the basis for plant determination according to Carl von Linné
Species are either plants or animals. Among animals we have the humans. Structures are structures over either a definite number of endurants of usually distinct sorts, or an indefinite number of endurants of the same sort.

3.5.3 Summary

The categorisation into structures, natural parts, artefactual parts, plants, and animals is thus partly based in Sørlander’s Philosophy, partly pragmatic. The distinction between endurants and perdurants, are necessitated by Kai Sørlander’s Philosophy as entities being in space, respectively entities being in space and time. Furthermore: discrete and continuous are motivated by arguments of natural sciences; structures are purely pragmatic; plants and animals, including humans, are necessitated by Kai Sørlander’s Philosophy. The distinction between natural, physical parts, and artefacts is not necessary in Sørlander’s Philosophy, but, we claim, necessary, philosophically, in order to perform the intentional “pull”, a transcendental deduction.

The distinction between part-materials, material-parts, and part-parts is pragmatic. We could have chosen another sub-ontology for artefacts. Also, from empirical observation, there seems to be no need for material-materials conjoins; In part-materials conjoins the materials are “contained within” the part; in material-parts conjoins the material, anthropomorphically speaking, “contains the parts”; and in part-parts conjoins the parts are monitored and controlled by the part. In a perceived material-materials artefact, should the material “contain” the materials? How?

3.5.3.1 Space and Time: Whither Entities?

Are space and time entities? Of course not! They are simply abstract concepts that apply to any entity.

3.6 Endurants and Perdurants

The concepts of endurants and perdurants are not present in, that is, are not essential to Sørlander’s Philosophy. Since our departure point is that of computing science where, eventually, conventional computing
performs operations on, i.e. processes data, we shall, however, introduce these two notions: endurant and perdurant. The former, in a rough sense, “corresponds” to data; the latter, similarly, to processes.

Philosophers have otherwise spent quite some thoughts on endurants and perdurants. It seems obvious that entities exist in space. But how do entities persist through time? Two accounts of persistence are endurance theory (endurantism) and perdurance theory (perdurantism). We shall basically stay clear of these, the footnoted sources, and rely on Kai Sørlander’s Philosophy.

Method Step 4 Initial Focus is on Endurants:

A basic principle of the domain analyser & describer method is that of initially focusing on endurants. Once all we wish to know about domain endurants has been analysed and described, then we shift focus to perdurants.

3.6.1 Endurants

Definition: 31 Endurant: By an endurant we shall understand an entity that can be observed, or conceived and described, as a “complete thing” at no matter which given snapshot of time; alternatively an entity is endurant if it is capable of enduring, that is persist, “hold out” [268, Vol. I, pg. 656]. Were we to “freeze” time we would still be able to observe the entire endurant.

Example 11 Geography Endurants: Geography endurants are: fields, meadows, lakes, rivers, forests, hills, mountains, et cetera. Railway System Endurants: a railway system, its net, its individual tracks, switch points, trains, their individual locomotives, et cetera.

A Caveat: Please observe the following: In Example 11 we seemingly rather easily refer to such things as fields, meadows, lakes, rivers, etc., as endurants that can be singled out from one another. It probably took mankind millenia to make this categorisation. Easier, perhaps, with the artefacts: railway net, track, locomotives, etc. These endurants were so designated by their designers, and we have kept these designations.

Analysis Predicate Prompt 2 is_endurant:

The domain analyser analyses an entity, \(\phi\), into an endurant as prompted by the domain analysis prompt:

- \(\text{is_endurant} – \phi\) is an endurant if \(\text{is_endurant}(\phi)\) holds.

is_entity is a prerequisite prompt for is_endurant.

3.6.2 Perdurants

Definition: 32 Perdurant: By a perdurant we shall understand an entity for which only a fragment exists if we look at or touch them at any given snapshot in time. Were we to freeze time we would only see or touch a fragment of the perdurant [268, Vol. II, pg. 1552].

Example 12 Geography Perdurants: the continuous changing of the weather (meteorology); the erosion of coast lines; the rising of some land and the “sinking” of other land areas; volcano eruptions; earthquakes; et cetera. Railway System Perdurants: the ride of a train from one railway station to another; and the stop of a train at a railway station from some arrival time to some departure time.

29 en.wikipedia.org/wiki/Formal_ontology#Endurant
30 https://en.wikipedia.org/wiki/Formal_ontology#Perdurant
31 plato.stanford.edu/entries/temporal-parts/
Analysis Predicate Prompt 3 is_perdurant:

The domain analyser analyses an entity $e$ into perdurants as prompted by the domain analysis prompt:

- \textbf{is_perdurant} -- $e$ is a perdurant if $\text{is\_perdurant}(e)$ holds.

is_entity is a prerequisite prompt for is_perdurant.

Occurrent, accident, continuant and happening are synonyms for perdurant.

We shall, in this monograph not develop an analysis calculus for perdurants, but leave such a, to us interesting research challenge to capable readers.

3.7 Endurants: Discrete and Continuous

We decide to facilitate the modelling of two kinds of endurants: discrete endurants and continuous endurants. Discrete endurants, we allow, may contain continuous endurants.

3.7.1 Discrete Endurants

\textbf{Definition:} 33 \textit{Discrete Endurant:} By a \textit{discrete endurant} we shall understand an endurant which is separate, individual or distinct in form or concept.

Analysis Predicate Prompt 4 is_discrete:

The domain analyser analyses endurants, $e$, into discrete entities as prompted by the domain analysis prompt:

- \textbf{is_discrete} -- $e$ is discrete if $\text{is\_discrete}(e)$ holds.

To simplify matters we shall allow separate elements of a discrete endurant to be continuous! That is, a discrete endurant, i.e., a part, may be conjoined with a continuous endurant, a material; we refer to Sect. 3.18.4 on Page 69.

\begin{center}
\textbf{Discrete Endurants}
\end{center}

\textbf{Example 13} The individual endurants of the above example of railway system endurants, Example 11 on the preceding page, were all discrete. Here are examples of discrete endurants of pipeline systems. A pipeline and its individual units: wells, pipes, valves, pumps, forks, joins, and sinks.

\textbf{Caveat:} Be aware of the following problem. Just because you ascribe the type name \textit{valve} to a discrete endurant, $e$, does not “automatically” endow the so-typed entity, $e$, with all, or at least some of those qualities that \textit{valve} values, such as you and I can agree on as being \textit{valve} values, do possess. No, we have to do much more analysis for “your” naming an entity of type \textit{valve}, and for that entity to indeed be what others would associate with \textit{valve} values. That “much more” analysis entails ascribing a sufficient number of internal qualities to what we labeled as valves, qualities such as unique identification, mereology and attributes.

3.7.2 Continuous Endurants: Materials

\textbf{Definition:} 34 \textit{Continuous Endurant:} By a \textit{continuous endurant} we shall understand an endurant which is prolonged, without interruption, in an unbroken series or pattern.

Analysis Predicate Prompt 5 is_continuous:
The domain analyser analyses endurants \( e \) into continuous entities as prompted by the domain analysis prompt:

- \( \text{is_continuous or is_material} \) – \( e \) is discrete if \( \text{is_continuous}(e) (\text{is_material}(e)) \) holds

We shall prefer to refer to continuous endurants as materials. Continuous materials are otherwise liquid, or gaseous, or plasmatic, or granular, or plant products, i.e., chopped sugar cane, threshed, or otherwise, et cetera.

**Materials**

**Example 14** Specific examples of materials are: water, oil, gas, compressed air, etc. A container, which we consider a discrete endurant, may be conjoined with another, now continuous endurant, a material, like a gas pipeline unit may “contain” gas. We refer to Sect. 3.18.4 on Page 69.

We cover materials further in Sect. 3.12 on Page 54.

Continuity shall here not be understood in the sense of mathematics. Our definition of ‘continuity’ focused on prolonged, without interruption, in an unbroken series or pattern. In that sense materials shall be seen as ‘continuous’. The mathematical notion of ‘continuity’ is an abstract one. The endurant notion of ‘continuity’ is physical one.

**Method Step 5 Discrete versus Continuous:**

One may question the distinction between discrete and continuous endurants. For most natural, God-given, and probably all man-made discrete endurants the temperature of its surroundings may decide its state of “firmness” or “fluidity”! We decide here to leave this, to some, crucial aspect untreated!

### 3.8 Discrete Endurants: Physical Parts, Structures and Living Species

We decide to analyse endurants into either of three kinds: physical parts, structures and living species. The distinction between the first two is pragmatic. The distinction between these and living species is motivated in Kai Sørlander’s Philosophy.

#### 3.8.1 Compound Endurants, “Roots” and “Siblings”

We need, in the following, to make definitions based on endurants being compounds.

**Definition: 35 Compound Endurants:** By a compound endurant we shall understand an endurant which can be considered as comprising two elements: a “root” and one or more, usually more, “siblings”. The “root” endurant, which is ignored for so-called endurant structures, can otherwise be said to “embody”, to “host”, the “siblings”. These, the “siblings”, can be said to be sub-ordinate to the “root”, also for endurant structures! These definitions may seem vague, but are in fact, sufficiently precise!

#### 3.8.2 Physical Parts

*Physical parts* are either *natural* parts, or are *artefactual* parts, i.e. man-made. Natural and man-made parts are either atomic or composite. We additionally analyse artefacts into conjoins, i.e., compounds of a “root” part and a definite number of different sort “sibling” materials, or a “root” material and an indefinite number of same-sort “sibling” parts, or a “root” part and an indefinite number of same-sort “sibling” parts.
Definition: Physical Parts: By a physical part we shall understand a discrete endurant existing in time and subject to laws of physics, including the causality principle and gravitational pull.

This characterisation is the result of our study of relations between philosophy and computing science, notably influenced by Kai Sørlander’s Philosophy. We refer to our research report [72, www.imm.dtu.dk/~dibj/2018/-philosophy/filo.pdf].

Analysis Predicate Prompt 6 is_physical_part: The domain analyser analyses “things” (e) into physical part. The method provides the domain analysis prompt:

• is_physical_part – where is_physical_part(e) holds if e is a physical part.

Physical parts are going to be the “workhorse” of our analyses & descriptions of artefactual domains.

Example 15 The example is that of the production of rum. From

1. the sowing, watering, and tending to of sugar cane plants;
2. via the “burning” of these prior to harvest;
3. the harvest;
4. the collection of harvest from sugar cane fields to the
5. the chipping, crushing, (and sometimes repeated) boiling, cooling and centrifuging of sugar cane when making sugar and molasses (into A, B, and low grade batches);
6. the fermentation, with water and yeast, producing a ‘wash’;
7. the (pot still or column still) distilling of the wash into rum;
8. the aging of rum in oak barrels;
9. the charcoal filtration of rum;
10. the blending of rum;
11. the bottling of rum;
12. the preparation of cases of rum for sales/export; and
13. the transportation away from the rum distiller of the rum.

Some comments on Example 15: Each of the enumerated items above is phrased in terms of perdurants. Behind each such perdurant lies some endurant. That is, in English, “every noun can be verbed”, and vice-versa. So we anticipate the transcendence, from endurants to perdurants.

Section 3.13 on Page 55 continues our treatment of physical parts.

3.8.3 Structures

Definition: Endurant Structure: By an endurant structure, or just , we shall understand a discrete endurant whose “root” element the domain engineer chooses to ignore, i.e., to not endow with internal qualities such as unique identifiers, mereology and attributes; but whose “siblings” are described as consisting of one or more discrete endurants.

Analysis Predicate Prompt 7 is_structure: The domain analyser analyses “things” (e) into structures. The method provides the domain analysis prompt:

• is_structure – where is_structure(e) holds if e is a structure.

We refer to Sect. 3.10 for further analysis of structures into set and composite structures.

3.8.4 Living Species

Living Species are either plants or animals. Among animals we have the humans.
3.9 Physical Parts: Natural Parts and Artefacts

We shall examine two kinds of physical parts: natural and man-made, i.e., artefactual, parts.

### 3.9.1 Natural Parts

**Definition:** 41 Natural Parts: Natural parts are not artefactuals, but are given by nature; are in space and time; are subject to the laws of physics, and also subject to the principle of causality and gravitational pull. Natural parts are parts which the domain engineer chooses to endow with all three internal qualities: unique identification, mereology, and one or more attributes.

**Analysis Predicate Prompt 9** is\_natural\_part: The domain analyser analyses “things” (e) into natural parts. The method provides the domain analysis prompt:

- is\_natural\_part – where is\_natural\_part(e) holds if e is a natural part.

**Example 17 Natural Parts: River Systems** Further examples of natural parts are: a river system – with its short or long stretches of water sources (springs, glaciers, meadows or even lakes) emerging into brooks or streams or rivers; winding or straight brook, stream and river sections; lakes; waterfalls; confluences of brooks, streams and rivers into appropriate ones of these; and divergences of either ones of these into appropriate ones of these.

### 3.9.2 Artefacts – Man-made Parts

**Definition:** 42 Man-made Parts: Artefacts: Artefacts are man-made either discrete or continuous endurants. In this section we shall only consider discrete endurants. Man-made continuous endurants are not treated separately but are lumped with natural materials. Artefacts are subject to the laws of physics, and are parts which, like for natural parts, the domain engineer chooses to endow with all three internal qualities: unique identification, mereology, and one or more attributes.

**Analysis Predicate Prompt 10** is\_artefact: The domain analyser analyses “things” (e) into artefact. The method provides the domain analysis prompt:

- is\_artefact – where is\_artefact(e) holds if e is an artefact.
Example 18  Artefactual Parts: Financial Service Industry  A further example of man-made parts are those of a financial service industry – taken here in a wide sense: (a) customers of any of the below; (b-d) banks: savings & loan, commercial and investment banks; (e) foreign exchange services; (f) insurance; (g-h) stock brokers and exchanges; (i) [other] commodities exchanges, (j) credit unions; (k) credit card companies; (l) accountancy companies; (m) consumer finance companies; (n) investment funds; and (o- . . . ) government and international overseeing agencies (national banks, The World Bank, International Monetary Fund (IMF), European Central Bank (ECB), etc).

We shall assume, cf. Sect. 4.4 [Attributes], that artefacts all come with an attribute of kind intent, that is, a set of purposes for which the artefact was constructed, and for which it is intended to serve.

3.10  Physical Parts: Structures

3.10.1  General

Structures are “conceptual, composite endurants”. A structure “gathers” one or more endurants under “one umbrella”, often simplifying a presentation of some elements of a domain description. Sometimes, in our domain modelling, we choose to model an endurant as a structure, sometimes as a composite part; it all depends on what we wish to focus on in our domain model. As such structures are “compounds” where we are interested only in the (external and internal) qualities of the elements, the “siblings” of the compound, but not in the qualities of the structure, i.e., the “root” itself.

Example 19  A transport system is modelled as structured into a road net structure and an automobile structure. The road net structure is then structured as a pair: a structure of hubs and a structure of links. These latter structures are then modelled as set of hubs, respectively links.

Structures versus Composites

Example 20  We could have modelled the road net structure as a composite part with unique identity, mereology and attributes which could then serve to model a road net authority. We could have modelled the automobile structure as a composite part with unique identity, mereology and attributes which could then serve to model a department of vehicles.

Whether to analyse & describe a discrete endurant into a structure or a physical part is a matter of choice. If we choose to analyse a discrete endurant into a physical part then it is because we are interested in endowing the part with internal qualities, the unique identifiers, mereology and one or more attributes. If we choose to analyse a discrete endurant into a structure then it is because we are not interested in endowing the endurant with qualities. When we choose that an endurant sort should be modelled as a part sort with unique identification, mereology and proper attributes, then it is because we eventually shall consider the part sort as being the basis for transcendentally deduced behaviours.

3.10.2  Compound Structures

Definition: 43  Compound Structure:  By a compound structure we shall understand a discrete endurant which the domain engineer chooses to describe as consisting of a definite number of discrete “sibling” endurants of usually distinct sorts but to not endow the “root” element with internal qualities such as unique identifiers, mereology and attributes.

Analysis Predicate Prompt 11  is compound structure:  The domain analyser analyses “things” (e) into compound structures. The method provides the domain analysis prompt:
3.11 Physical Parts: Living Species – Plants and Animals

We refer to Sect. 3.8.4 for our first characterisation (Page 51) of the concept of living species\textsuperscript{32}: a discrete endurant existing in time, subject to laws of physics, and additionally subject to causality of purpose.\textsuperscript{33}

\textbf{Definition: 45 Living Species, II:} Living species must have some form they can be developed to reach; which they must be causally determined to maintain. This development and maintenance must further engage in an exchange of matter with an environment. It must be possible that living species occur in one of two forms: one form which is characterised by development, form and exchange; another form which, additionally, can be characterised by the ability to purposeful movement. The first we call plants, the second we call animals.

It is appropriate here to mention Carl Linnaeus (1707–1778). He was a Swedish botanist, zoologist, and physician who formalised binomial nomenclature, the modern system of naming organisms. He is known as the “father of modern taxonomy”. We refer to http://www.gutenberg.org/ebooks/20771.

3.11.1 Plants

\textbf{Example 21} Although we have not yet come across domains for which the need to model the living species of plants were needed, we give some examples anyway: grass, tulip, rhododendron, oak tree.

\textbf{Analysis Predicate Prompt 13 is\_plant:} The domain analyser analyses “things” ($\ell$) into a plant. The method provides the domain analysis prompt:

\begin{itemize}
  \item \textbf{is\_plant} – where \textit{is\_plant}($\ell$) holds if $\ell$ is a plant
\end{itemize}

The predicate \textit{is\_living\_species}($\ell$) is a prerequisite for \textit{is\_plant}($\ell$).

\textsuperscript{32} See analysis prompt 8 on Page 51.
\textsuperscript{33} See Footnote a on Page 50.
DOMAINS – A Taxonomy

External Qualities

3.11.2 Animals

**Definition:** 46 Animal: We refer to the initial definition of living species above – while emphasizing the following traits: (i) a form that animals can be developed to reach and (ii) causally determined to maintain through (iii) development and maintenance in an exchange of matter with an environment, and (iv) ability to purposeful movement.

**Analysis Predicate Prompt 14 is_animal:** The domain analyser analyses “things” (ℓ) into an animal. The method provides the domain analysis prompt:

- **is_animal** – where **is_animal(ℓ)** holds if ℓ is an animal.

The predicate **is_living_species(ℓ)** is a prerequisite for **is_animal(ℓ)**.

Example 22 Although we have not yet come across domains for which the need to model the living species of animals, in general, were needed, we give some examples anyway: A band of musicians, a swarm of flies, a bunch of crooks, a crew of sailors, a gang of outlaws, a group of people, a herd of cattle, a mob of hair, a pack of dogs, a flock of geese, a pride of lions, and a school of dolphins.

3.11.2.1 Humans

**Definition:** 47 Human: A human (a person) is an animal, cf. Definition 46, with the additional properties of having language, being conscious of having knowledge (of its own situation), and responsibility.

**Analysis Predicate Prompt 15 is_human:** The domain analyser analyses “things” (ℓ) into a human. The method provides the domain analysis prompt:

- **is_human** – where **is_human(ℓ)** holds if ℓ is a human.

The predicate **is_animal(ℓ)** is a prerequisite for **is_human(ℓ)**.

We refer to [72, Sects. 10.4–10.5] for a specific treatment of living species, animals and humans, and to [72] in general for the philosophy background for rationalising the treatment of living species, animals and humans.

We have not, in our many experimental domain modelling efforts had occasion to model humans; or rather: we have modelled, for example, automobiles as possessing human qualities, i.e., “subsuming humans”. We have found, in these experimental domain modelling efforts that we often confer anthropomorphic qualities on artefacts, that is, that these artefacts have human characteristics. You, the reader are reminded that when some programmers try to explain their programs they do so using such phrases as *and here the program does ... so-and-so!*

3.12 Continuous Endurants: Materials

**Definition:** 48 Material: By a material we shall understand a continuous endurant.

We shall simplify our treatment of materials. We model a material as potentially consisting of an amalgam of one or more substances of different sorts. So a continuous endurant is a “single” material. Composite

---

34 Attributes of that “single” material may then reveal how it is (chemically or otherwise) composed from distinct substances.
physical parts may be conjoined with materials: natural parts may “contain” natural and artefactual materials, artefacts may “contain” natural and artefactual materials. We leave it to the reader to provide analysis predicates for natural and artefactual “materials”.

Example 23 A natural part, say a land area, may contain glaciers, springs, rivers, lakes, and border seas. An artefact, say an automobile, usually contains gasoline, lubrication oil, engine cooler liquid and window screen washer water.

Material substances are either liquid, like water, sewage, or oil; or gaseous, like natural gas; or plasmatic – a combination of granular and liquid forms, like blood; or granular, like iron ore, sand, or pebbles (stones, etc.); or agricultural, like sugar cane, chopped wood, grain, etc.

We refer to Sect. 3.18.5 on Page 73.

3.13 Natural Parts and Artefacts: Atomic, Composite and Conjoins

3.13.1 Atomic and Compound Parts

A distinguishing quality of natural and artefactual parts is whether they are atomic (Sect. 3.13.2) or compound (Sects. 3.13.3–3.13.4). Please note that we shall, in the following, examine the concept of parts in quite some detail. This is a choice. The choice is based on pragmatics. It is still the domain analyser cum describers’ choice whether to consider a discrete endurant a part or a structure. If the domain engineer wishes to investigate the details of a discrete endurant then the domain engineer chooses to model the discrete endurant as a part.

Compound Parts: Compound parts are analysed, we suggest, into composite endurants (Sect. 3.13.3 on the next page) and conjoins. Conjoins, (Sect. 3.13.4 on the following page), are further analysed into: material-parts conjoins (Sect. 3.13.4 and Pages 64, 64 and 70), part-materials conjoins (Sect. 3.13.4 and Pages 64 and 71), and part-parts conjoins (Sect. 3.13.4 and Pages 65 and 72). The term ‘compound’ is thus used to encompass four distinct categories of discrete endurants. ‘Composite’, as a term, is, to us, the more general concept and one could probably make-do with just that concept and not need the conjoin parts concepts. So our choice is one of pragmatics. It would sometimes be awkward to model endurants without the facility of concrete sets; and, using the conjoin modelling concept reveals intention! We shall have more to say about this in time.

Example 24 Before actually characterising the five categories of parts (atomic, composite, part-materials conjoins, material-parts conjoins and part-parts conjoins) we shall here hint at some examples. In modelling certain domains, given some further unspecified context, we may choose to model consumers, retailers, wholesalers and consumer product manufacturers as atomic, while the market is modelled as a part-parts conjoin of sets of consumers, retailers, wholesalers and consumer product manufacturers. In some other context we may choose otherwise! Along another line wells, pipes, pumps, valves, forks joins and sinks of an oil pipeline system are individually modelled as part-materials conjoins and the oil pipeline system as a composite of sets of these part-materials conjoins – each part-materials conjoin consisting of the overall part-materials conjoin, an atomic part and a definite set of materials of different sorts. Similarly for canal systems, waste management, rum production and water management systems (as in The Netherlands). And finally we may model, as a material-parts conjoin, air traffic as a single material-parts conjoin consisting of an atomic part (say the air traffic monitor & advisory authority) and a concrete set of distinct aircraft parts. Similarly for ocean ship monitor & advisory authorities et cetera.

35 We use the term ‘to model’ interchangeably with the phrase ‘to analyse & describe’; similarly a model is used interchangeably with an analysis & description.
3.13.2 Atomic Parts

**Definition:** Atomic Parts: Atomic parts are those which, in a given context, are deemed to not consist of meaningful, separately observable proper sub-parts. A sub-part is a part.

We emphasize the term ‘demeed’. The domain analyser & describer is the one who is ‘deeming’. It is all a choice.

**Analysis Predicate Prompt 16** is_atomic:

The domain analyser analyses a discrete endurant, i.e., a part \( p \) into an atomic endurant:

- **is_atomic:** \( p \) is an atomic endurant if is_atomic\( (p) \) holds.

is_discrete is a prerequisite prompt of is_atomic.

The is_atomic analysis prompt comes in two variants: is_natural_atomic and is_artefactual_atomic. Similarly for the is_composite analysis prompt: is_natural_composite and is_artefactual_composite. In the following we shall often omit the infix _natural_ or _artefactual_.

---

### Atomic Road Net Parts

**Example 25** From one point of view all of the following can be considered atomic parts: hubs, links, and automobiles.

---

3.13.3 Composite Parts

**Definition:** Composite Parts: Composite parts are those which, in a given context, are deemed to meaningfully consist of separately observable a "root" part and a definite number of proper "sibling" sub-parts of distinct sorts.

We emphasize the term ‘demeed’. The domain analyser & describer is the one who is ‘deeming’. It is all a choice.

**Analysis Predicate Prompt 17** is_composite:

The domain analyser analyses a discrete endurant, i.e., a part \( p \) into a composite endurant:

- **is_composite:** \( p \) is a composite endurant if is_composite\( (p) \) holds.

is_discrete is a prerequisite prompt of is_composite.

---

### Composite Automobile Parts

**Example 26** We refer to Example 25. From another point of view all of the following can be considered composite parts: an automobile, consisting of, for example, the following parts: the engine train, the chassis, the car body, the doors and the wheels. These can again be considered composite parts.

---

36 Hub \( \equiv \) street intersection; link \( \equiv \) street segments with no intervening hubs.
3.13.4 Conjoins

We suggest three kinds of conjoins: part-materials conjoins, material-parts conjoins and part-parts conjoins. We have decided to include these three endurant categories for the following reason: their use, the fact that the domain analyser & describer chooses to model a concept as a conjoin, shall reveal an intent. The intents are these. For part-materials conjoins and part-parts conjoins the two elements of the conjoin serve two very related, i.e., conjoined, rôles: (a) the part as the overall monitor (and potential controller), (b) the materials or parts as that which is being monitored and to some extent controlled by the part. For material-parts conjoins the two elements (the material, respectively the parts) serve two related, i.e., conjoined, rôles: (a) the material as the “carrier” of the (b) the parts whose raison d’être is that they can “inhabit” the material. We shall model the material as if it was an atomic part but such that it ‘embodies’ the material.

Analysis Predicate Prompt 18 is_conjoin:
The domain analyser analyses endurants $e$ into conjoin entities as prompted by the domain analysis prompt:

- is_conjoin – $e$ is a conjoin if is_conjoin($e$) holds

3.13.5 Part-Materials Conjoins

Definition: Part-Materials Conjoin: By a part-materials conjoin we shall understand an endurant which is a composition of a [“root”] part, and one or more (intentionally) distinct [“sibling”] materials.

The pragmatics of part-materials conjoins is that they serve to model such domains as water & flood management – as in The Netherlands State Water Management Authority[37]; canal systems, i.e., artefactual waterways, typically with locks – as in, for example the Panama Canal Authority[38]; water, oil, gas and other pipelines – as in, for example the (now defunct) Nabucco West Pipeline Proposal[39]; waste management – as in the European Union Waste Management Project[40]; uni-flow production systems – as in, for example, the production of spirits, like whiskey, rum, etc., and in industrial manufacturing. Where road transport nets typically be modelled as bi-directed, cyclic graphs, the above ‘conjoin nets’ typically can be modelled as directed, acyclic graphs[41]. A difference between water & flood management and canal systems is that the former primarily manages the water level, by means of pumps, whereas the latter primarily manages the passage of anywhere from [300 tons] barges to [300,000 tons] vessels, by means of locks.

Caveat: We could have stipulated that conjoins consist of one or more [“root”] physical parts, etc., but it appears, from modelling experience, that to settle on exactly one makes modelling “easier”!

Analysis Predicate Prompt 19 is_part_materials_conjoin:
The domain analyser analyses endurants $e$ into part-materials conjoin entities as prompted by the domain analysis prompt:

- is_part_materials_conjoin – $e$ is a part-materials conjoin if is_part_materials_conjoin($e$) holds

[37] www.rijkswaterstaat.nl/english/index.aspx
[38] www.pancanal.com/eng
[40] www.urban-waste.eu/project
[41] – although waste management systems may contain some cyclicity
We emphasize that the domain analyser & describer is making a choice. The domain analyser & describer is the one who ‘chooses’. The context and aims of the domain modelling effort decides which choices to make.

### Part-Material Conjoins: Pipelines, I

**Example 27** A pipeline consists of a number of conjoined pipeline units: The pipeline units (each with their “container” and some liquid). A pipeline unit is either a well (with some, zero, or a maximum of liquid), a pump, pumping or not (with some, zero, or a maximum of liquid), a pipe (with some, zero, or a maximum of liquid), a valve, closed or partially or fully open (with some, zero, or a maximum of liquid), a fork diverting a line into two (with some, zero, or a maximum of liquid) and a join merging two lines into one (with some, zero, or a maximum of liquid), a sink (with some, zero, or a maximum of liquid). Liquid flows in one direction, from wells to sinks. There are no cycles.

### Part-Material Conjoins: Canals with Locks, I

**Example 28** A system of canals with locks consists of a number of canal units. A canal unit is a conjoined endurant consisting of a pair: a discrete canal unit and a material – some “[muddy]” water! Canal units serve to convey canal vessels (pleasure boats, barges) in either direction of the canal system. Discrete canal units are either linear (or, for that matter a curved) stretches of a canal; fork/join: diverting/joining one stretch of canal into two, respectively two into one – forks/joins connect linear canal units whose water levels “agree”; or lock sequences of one or more single locks. A lock sequence connects two linear canal units whose water levels “disagree”. A single lock allows canal vessels to be lowered/raised in order to be conveyed into next single lock or linear canal units. Single locks may either be open in one or in the opposite direction, for a vessel in the single lock to sail out of the single lock in that direction or into the single lock from the opposite direction, or they may be closed, that is, in the process of lowering or raising its water level. Etc., etc.

### Part-Materials Conjoins: Waste Management, I

**Example 29** Waste management [systems] are about the transport and treatment of waste. Waste can, for example, be non-clean water, sewage, chemical side-products, or other. Transport can, for example, be pipes, barrels, conveyor belts, or other. Treatment can, for example, be removing undesired materials from non-clean water, sewage and chemicals resulting in at least clean water or desired chemical and one or more waste products. A waste treatment system consists, typically, of a number of conjoin units: sources of the waste, waste transport networks, where some segments of these networks converge on treatment plants, from which emerges two or more waste and non-waste networks.

In this monograph we shall exemplify excerpts of many different kinds of (of a category of) domains. A sub-category is that of domains primarily “populated” with conjoins such as those listed in the main opening, the pragmatics, paragraph of this section. We refer to Sect. 3.18.4 on Page 69, Example 40 on Page 71, Sect. 4.3.7 on Page 93, Sect. 4.4.8 on Page 108 and Sect. 6.10.1 on Page 152.

### 3.13.6 Material-Parts Conjoins

**Definition:** Material-Parts Conjoin: By a material-parts conjoin we shall understand an endurant which is a composition of the material-parts conjoin ["root"] material and zero, one or more (intentionally) distinct ["sibling"] parts.

The pragmatics of material-parts conjoins is that they serve to model such domains as: Vessel traffic on the open seas: the open seas are the material, i.e., the oceans, seas, great lakes and channels; vessels are the parts, they ply from harbour to harbour and sometimes on canals. Air traffic in the sky: the air space is the material; aircraft of all kinds are the parts.
Analysis Predicate Prompt 20 \textit{is\_material\_parts\_conjoin}:

The domain analyser analyses endurants $e$ into material-part conjoin entities as prompted by the \textit{domain analysis prompt}:

- \textit{is\_material\_parts\_conjoin} – $e$ is a material-parts conjoin if \textit{is\_material\_parts\_conjoin}(e) holds.

We again emphasize that the domain analyser & describer is making a choice. The domain analyser & describer is the one who is ‘chose’. The context and aims of the domain modelling effort decides which choices to make.

**Example 30** There is the \textit{material-parts conjoin} of air traffic containing an, or the, \textit{material} of air space, an \textit{EXTENT} in the sense of Sect. 2.4 on Page 22, with the \textit{parts} being a \textit{set} of aircraft.

**The Shipping Lanes**

**Example 31** There is the \textit{material-parts conjoin} of shipping lanes containing an, or the, \textit{material} of waterways, a \textit{SURFACE} in the sense of Sect. 2.4 on Page 22, with the \textit{parts} being a \textit{set} of vessels.

3.13.7 Part-Parts Conjoins

Part-parts conjoins come in two forms: (i) “proper” part-parts which embody a “root” and zero one or more, but a definite number of “siblings; and (ii) a “simplified” part-parts where we ignore the “root”. It is up to the domain analyser cum describer to make the choice whether to include the “root” or not. You may think of the latter as representing a structure (the “root”) with sets of “siblings”.

Analysis Predicate Prompt 21 \textit{is\_part\_parts\_conjoin}:

The domain analyser analyses endurants $e$ into part-parts conjoin entities as prompted by the \textit{domain analysis prompt}:

- \textit{is\_part\_parts\_conjoin} – $e$ is a part-parts conjoin if \textit{is\_part\_parts\_conjoin}(e) holds.

**Part-Parts Conjoin: Container Terminal Ports**

There is the set of \textit{composite} \textit{container terminal ports}.

There is the \textit{composite part} of the port itself, with its \textit{structures} of \textit{sets} of container vessels, \textit{structures} of \textit{sets} of quay side ship/quay cranes, \textit{structures} of \textit{sets} of quay crane to terminal port bay trucks, \textit{structures} of \textit{sets} of terminal port bays and \textit{structures} of \textit{sets} of container land trucks. The \textit{containers} are embodied in vessel bays, on quay cranes, on bay trucks, in terminal port bays and on land trucks.

There are a set of zero, one or more \textit{container vessels}: each vessel being a \textit{part-parts conjoin} of the “bare” vessel, containing a \textit{part set} of vessel \textit{container bays}.

There is a set quay cranes, each quay crane being a \textit{part-parts conjoin} of the “bare” crane hosting a \textit{part set} of zero or one container part.

---

42 While any stretch of water which is regularly frequented by ships can be called a shipping lane, it is more meaningful to limit the use of the term to real \textit{fairways}, in which shipping is limited by the depth of the water or by other navigational hazards and which normally – though not always – are marked by navigational aids. Thus the development of shipping lanes mainly involves navigation to and from ports and through shallow seas or bays, archipelagos, and narrow straits [www.encyclopedia.com].
There is a set quay crane to terminal port bay trucks, each bay truck being a part-parts conjoin of the “bare” truck hoisting a part set of zero, one or two containers.

There is a set of (vessel or terminal port) container bays, with each container bay being a structure container rows, with a container row being a structure container stack, with a container stack being a sequence containers.

There is a set of terminal port to and from customer land trucks, each such land truck being a part-parts conjoin of the “bare” truck hoisting a part set of zero or one container.

We refer to [73, Container Terminals, September 2018], an experimental case study report where we used this, the atomic, composite and concrete set approach. We refer to [376, A Unified Theory of Programming approach for rHiMo] an extension of CSP, with real-time and process mobility expressivity as a promising approach.

Method Step 6 From Analysis to Description:

We have reached a stage in our unraveling an, or the, analysis calculus where it is now possible to “switch” to a, or the, description calculus. That is, here is a step of the method: to conscientiously apply description prompts. These follow in Sect. 3.18.

To prepare for the external qualities description calculi we must, however, first review how we discover endurant sorts, Sect. 3.14, examine a notion of states, Sect. 3.15, review the unfolding ontology of endurants, Sect. 3.16 and introduce some endurant analysis functions (not predicates), Sect. 3.17.

3.14 On Discovering Endurant Sorts

The subject of ‘discovery’ depends very much on whether the endurant is an artefact or a natural part.

3.14.1 On Discovering Man-made Endurants

Artefacts are man-made. Usually the designers – the engineers, the craftsmen – who make these parts start out by ascribing specific names to them. And these names become our sort names. So the $\alpha, \beta, \gamma$ points below are really only relevant for the analysis of natural discrete endurants.

3.14.2 On Discovering Natural Endurants

Our aim now is to present the basic principles that let the domain analyser decides on endurant sorts. We observe endurants one-by-one.

$(\alpha)$ Our analysis of parts concludes when we have “lifted” our examination of a particular endurant instance to the conclusion that it is of a given sort, that is, reflects a formal concept.

Thus there is, in this analysis, a “eureka”, a step where we shift focus from the concrete to the abstract, from observing specific endurant instances to postulating a sort: from one to the many. If $e$ is an endurant of sort $E$, then we express that as: $e : E$.

Analysis Function Prompt 1 analyse_endurant_sorts:

The domain analyser analyses composite endurants, whether natural or man-made, into a definite set of endurants and their sort names. The method provides the domain analysis prompt:

- analyse_endurant_sorts directs the domain analyser to observe a definite set of sibling endurants and their sort names:

$$\text{value analyse_endurant_sorts}(e) \equiv \{e_1, \ldots, e_n\}, (\alpha E_1, \ldots, E_n)$$
The analyse_endurant_sorted function is meta-linguistic, that is, informal. When we say that analyse_endurant_sorted(e) = \{e_i=E_1,E_2,\ldots,E_m\} we mean we have observed m distinct endurant values, e_i, and their sort names, \(E_i\) [i: 56.18].

In Sect. 3.17 we shall introduce analysis functions for all the endurants of our ontology.

**Elaboration 1 Type, Values and Type Names:** This is the first time we shall be quoting, in this instance, RSL-Text type names.

\[(\beta)\] The analyser analyses, for each of these endurants, \(e_i\), which formal concept, i.e., sort, it belongs to; let us say that it is of sort \(E_k\); thus the sub-parts of \(e\) are of sorts \(\{E_1,E_2,\ldots,E_m\}\). Some \(E_k\) may be natural parts, other artefacts, or structures, or materials. And parts may be either atomic or composite.

The domain analyser continues to examine a finite number of other composite parts: \(\{p_1,p_2,\ldots,p_n\}\). It is then “discovered”, that is, decided, that they all consists of the same number of sub-parts \(\{e_{i_1},e_{i_2},\ldots,e_{i_n}\}, \{e_{j_1},e_{j_2},\ldots,e_{j_n}\}\), \(\ldots\), \(\{e_{m_1},e_{m_2},\ldots,e_{m_n}\}\), of the same, respective, endurant sorts.

\[(\gamma)\] It is therefore concluded, that is, decided, that \(\{e_1,e_2,\ldots,e_n\}\) are all of the same endurant sort \(E\) with observable part sub sorts \(\{E_1,E_2,\ldots,E_m\}\).

Above we have type-font-highlighted three sentences: (\(\alpha, \beta, \gamma\)). When you analyse what they “prescribe” you will see that they entail a “depth-first search” for endurant sorts. The \(\beta\) sentence says it rather directly: “The analyser analyses, for each of these endurants, \(p_k\), which formal concept, i.e., endurant sort it belongs to.” To do this analysis in a proper way, the analyser must (“recursively”) analyse structures into sub-structures, parts and materials, and parts “down” to their atomicity. For the parts (whether natural or man-made) and materials of structures the analyser cum describer decides on their sort, and work (“recurse”) their way “back”, through possibly intermediate endurants, to the \(p_1\)’S. Of course, when the analyser starts by examining atomic parts and materials, then their endurant structure and part analysis “recursion” is not necessary.

Thus the discovery of natural parts and natural materials is very much of the kind that the Swedish 18th century botanist, zoologist, and physician Carl von Linné (Carl Linnaeus, 1707–1778) who formulated the so-called binomial nomenclature, the system of naming organisms. Linné is also referred to as “the father of taxonomy”.

**3.15 States**

In our continued modelling we shall make good use of a concept of states.

**Definition:** By a state we shall understand any collection of one or more parts.

In Chapter 4 we introduce the notion of attributes. Among attributes there are the dynamic attributes. They model that internal part quality values may change dynamically. So we may wish, on occasion, to “refine” our notion of state to be just those parts which have dynamic attributes.

Given any universe of discourse, uod:UoD, we can recursively calculate its “full” state.

104 Let \(\theta\) be any endurant.

Let arg_parts be the parts to be calculated.

Let res_parts be the parts calculated.

Initialise the calculator with arg_parts={\(\theta\)} and res_parts={}.

Calculation stops with arg_parts empty and res_parts the result.

105 If is_composite(\(\theta\)) then we obtain its immediate parts, analyse_composite_part(\(\theta\));

now rearrange argument and result parameters:

remove \(\theta\) from in_parts;

add analyse_composite_part(\(\theta\)) to in_parts; and

join \(\theta\) to out_parts.

106 If is_part_parts(\(\theta\)) then we obtain its immediate parts;

then suitably rearrange argument and result parameters.
If `is_part_materials(∅)` then we obtain its immediate parts; then suitably rearrange argument and result parameters. And so forth!

```
value
104. calc_parts: E-set → E-set → E-set
104. calc_parts(arg_parts)(res_parts) ≡
104. if arg_parts = ∅ then res_parts else
104. let e ⋅ e ∈ arg_parts in
105. isComposite(e) →
105. calc_parts(analyseCompositePart(e))(res_parts ∪ {e})
106. isPart(e) →
106. calc_parts(analysePartParts(e))(res_parts ∪ {e})
107. isPartMaterials(e) →
107. calc_parts(analysePartMaterials(e))(res_parts ∪ {e})
108. et cetera!
104. end end
```

### Constants and States

**Example 33**

109 Let there be given a universe of discourse, `rt s`. It is an example of a state.

From that state we can calculate other states.

110 The set of all hubs, `hs`.
111 The set of all links, `ls`.
112 The set of all hubs and links, `hls`.
113 The set of all bus companies, `bcs`.
114 The set of all buses, `bs`.
115 The set of all private automobiles, `as`.
116 The set of all parts, `ps`.

```
value
109 rt s:UoD [109]
110 hs:H-set ≡ obsSH(obsSH(obsRN(rt s)))
111 ls:L-set ≡ obsSL(obsSL(obsRN(rt s)))
112 hls:(H|L)-set ≡ hs∪ls
113 bcs:BC-set ≡ obsBCs(obsSBC(obsFV(obsRN(rt s))))
114 bs:B-set ≡ {obsBs(bc)|bc:BCs(bc ∈ bcs}
115 as:A-set ≡ obsBCs(obsSBC(obsFV(obsRN(rt s))))
116 ps:(UoB|H|L|BC|B|A)-set ≡ rt s∪hls∪bcs∪bs∪as
```

### Method Step 7  Domain State:

We have found, once all the state components, i.e., the endurant parts, have had their external qualities analysed, that it is then expedient to define the domain state. It can then be the basis for several concepts of internal qualities.

We refer to Sect. 6.2.1 on Page 125 for more on states.
3.16 A Review of the Ontology of Endurants

It is time to review the ontology of endurants. We refer to Figure 3.1 on Page 44. Black circles, ●, designate a category of entities. There are 21 such endurant categories. Two or three slanted lines connect entity category black circles ●. Where no lines emanate from (13 of) of these ●s it means that the endurants are either atomic (p:A), or a composite of endurants ((e₁:E₁, e₂:E₂, ..., eₙ:Eₙ}), or a concrete set of endurants ({e₁, e₂, ..., eₙ}), or a conjoin, or a material (m:M) – such as so labelled in Figure 3.1 on Page 44. In all other cases 2 or 3 lines emanates from the other ●s. They indicate, for an endurant value of the bullet labelled category, that there are two, respectively three or four choices naming disjoint endurant sorts. In Fig. 3.3 we have, as an example, labelled the two down-ward emanating edges from the physical_parts bullet, with the analysis prompt names corresponding to the endurant categories upon which they are incident. We represent Fig. 3.1 on Page 44 in Fig. 3.3 emphasizing the above points.

Fig. 3.3. The Endurant Analysis Hierarchy

The table above lists the full ensemble of analysis prompts (several of which will only be covered later). To connect the presentation of analysis prompts, now, to the presentation of description prompts, we have added five square magenta-coloured boxes, ■, to Figure 3.3. They designate the five “analysis states” at, or in, which we can apply corresponding description prompts. That is, we have built up all the necessary analysis issues and are ready to “take the consequences”, that is, to draw the necessary conclusions, given such-and-such an endurant category we can inquire about and describe its sorts and their observer functions.
3.17 Endurant Analysis Function Prompts

We need informally define some analysis functions to be used in the domain description prompt definitions; one, basically, for each non-atomic endurant sort.

Analysis Function Prompt 2 \texttt{analyse}\_\texttt{compound}\_\texttt{structure}\_\texttt{sorts}:

The domain analyser analyses structures into the siblings of a compound structure, i.e., an indefinite set of distinct sort endurants and their sort names. The method provides the \textit{domain analysis prompt}:

- \texttt{analyse}\_\texttt{compound}\_\texttt{structure} directs the domain analyser to observe the sibling .

\begin{equation}
\text{value}\ \text{analyse}\_\texttt{compound}\_\text{structure}\_\texttt{sorts}(e) \equiv \{(e_1,\ldots,e_n),(\cdot \cdot \cdot E_i \ldots E_n \cdot \cdot \cdot)\}
\end{equation}

Analysis Function Prompt 3 \texttt{analyse}\_\texttt{set}\_\texttt{structure}\_\texttt{sort}:

The domain analyser analyses structures into the siblings of a set structure, i.e., an indefinite set of same sort endurants and their common sort name. The method provides the \textit{domain analysis prompt}:

- \texttt{analyse}\_\texttt{set}\_\texttt{structure}\_\texttt{sort} directs the domain analyser to observe the sibling .

\begin{equation}
\text{value}\ \text{analyse}\_\texttt{set}\_\text{structure}\_\texttt{sort}(e) \equiv \{(p_1,\ldots,p_n),(\cdot \cdot \cdot P \cdot \cdot \cdot)\}
\end{equation}

Analysis Function Prompt 4 \texttt{analyse}\_\texttt{material}\_\texttt{parts}\_\texttt{material}:

The domain analyser analyses material-parts conjoins into a material and its type name. The method provides the \textit{domain analysis prompt}:

- \texttt{analyse}\_\texttt{material}\_\texttt{parts}\_\texttt{material} directs the domain analyser to observe the material.

\begin{equation}
\text{value}\ \text{analyse}\_\texttt{material}\_\text{parts}\_\texttt{material}(e) \equiv m:(\cdot \cdot \cdot M \cdot \cdot \cdot)
\end{equation}

\textbf{Elaboration 2 Type, Values and Type Names:} The endurant analysis functions, this and the below, all illustrate quoting .

\begin{center}
\textbf{Material of an Airspace}
\end{center}

\textbf{Example 34} The material of an air space is that airspace, an EXTENT.\textsuperscript{44}

\begin{center}
\textbf{Parts of an Airspace}
\end{center}

\textbf{Example 35} The parts of an Airspace is a set of zero, one or more aircraft.

\textsuperscript{44} See Item 69 on Page 23 of Sect. 2.4 on Page 22.
3.18 Endurant Describers

Based on the analyses of Sects. 3.5.3.1–3.13, we conclude that there are six kinds of endurants to sort (i.e., type) and observer function describe:

### Analysis Function Prompt 6  `analyse_part MATERIALS MATERIALS`

The domain analysis prompt

- `analyse_part MATERIALS MATERIALS` directs the domain analyser to observe a “compound” of one or more materials that the conjoin embodies – together with their material sort names.

  \[\text{value} \quad \text{analyse}_\text{part MATERIALS MATERIALS}(e) \equiv (\langle m_1, m_2, \ldots, m_m \rangle, \langle \mathit{M}_1, \ldots, \mathit{M}_m \rangle)\]

---

### Example 36

The materials of a wastewater management system treatment conjoin are several: the sewage, water, carbon dioxide, water, bio-solids, etc.

---

### Analysis Function Prompt 7  `analyse_part MATERIALS PART`

The domain analysis prompt

- `analyse_part MATERIALS PART` directs the domain analyser to observe a part-materials conjoin for the “root” part and its sort:

  \[\text{value} \quad \text{analyse}_\text{part MATERIALS PART}(e) \equiv p: \mathit{P}\]

---

### Analysis Function Prompt 8  `analyse_part PARTS PARTS`

The domain analysis prompt

- `analyse_part PARTS PARTS` directs the domain analyser to observe a set of zero, one or more parts that the conjoin embodies – together with their (single) part sort name.

  \[\text{value} \quad \text{analyse}_\text{part PARTS PARTS}(e) \equiv \{p_1, p_2, \ldots, p_m\}: \mathit{P}\]

---

### Example 37

The parts of links, respectively hub aggregates, i.e., part-parts conjoins, are a set of links, respectively a set of hubs of the road net.

---

### Analysis Function Prompt 9  `analyse MATERIAL`

The domain analyser analyses material into that material’s value and type name. The method provides the domain analysis prompt:

- `analyse MATERIAL` directs the domain analyser to observe the material.

  \[\text{value} \quad \text{analyse}_\text{MATERIAL}(e) \equiv m: \mathit{M}\]

---

### 3.18 Endurant Describers

Based on the analyses of Sects. 3.5.3.1–3.13, we conclude that there are six kinds of endurants to sort (i.e., type) and observer function describe:
domains – an taxonomy

eXternal qualities

• composite parts,
• structures,
• conjoins – in three variants – and
• materials.

Atomic parts are what is left, when composites, structures and conjoins have no further sub endurants. The
general signature of the describer functions are of the form:

\[ \text{value} \text{ describe}_* : E \rightarrow \text{RSL-Text} \]

3.18.1 Describing Composite Parts

The above analysis amounts to the analyser first “applying” the domain analysis prompt \( \text{is\_composite}(e) \) to a discrete endurant, \( e \), where we now assume that the obtained truth value is true. Let us assume that endurants \( e:E \) consist of sub-endurants of sorts \( \{E_1,E_2, \ldots, E_m\} \). Since we cannot automatically guarantee that our domain descriptions secure that \( E \) and each \( E_i (1 \leq i \leq m) \) denotes disjoint sets of entities we must prove it.

Domain Description Prompt 2 \( \text{describe\_endurant\_sorts} \): If \( \text{is\_composite}(e) \) holds, then the analyser “applies” the domain description prompt

• \( \text{describe\_endurant\_sorts}(e) \)

resulting in the analyser writing down the endurant sorts and endurant sort observers domain description text according to the following schema:

\[ 2. \text{describe\_endurant\_sorts}(e) \]

<table>
<thead>
<tr>
<th>[ \text{Describer} ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>let ( { \text{type}<em>{E_1}, \ldots, \text{type}</em>{E_m} } = \text{analyse_endurant_sorts}(e) ) in</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

The use of the underscore, \_ shall inform the reader that there is not need, here, for naming a value.

Elaboration 3 Type, Values and Type Names: Note the use of quotes above. Please observe that when we write \( \text{obs}\_E \) then \( \text{obs}\_E \) is the name of a function. The \( E \), when juxtaposed to \( \text{obs}\_ \) is now a name.

Analysis Function Prompt 10 type name, type of, is_:

The definition of \( \text{obs}\_E \) implicitly implies the definition of

\[ \text{obs}_E(e) = e_i \supset \text{type}\_name(e_i) = E_i \land \text{type}\_of(e_i) = E_i \land \text{is}\_E(e_i) \]
3.18.2 Describing Compound Structures

Recall, from Sect. 3.10.2, that there are two kinds of structural endurants: compound and set. We first treat the compound structures.

Domain Description Prompt 3 describe\textunderscore compound\textunderscore structure\textunderscore sorts: If \textit{is\_compound\_structure}(e) holds, then the analyser “applies” the \textit{domain description prompt}

\begin{itemize}
  \item \texttt{describe\_compound\_structure\_sorts(e)}
\end{itemize}

resulting in the analyser writing down the \textit{compound structure sorts} and \textit{compound structure sort observers}

\begin{itemize}
  \item \texttt{domain description text according to the following schema:}
\end{itemize}

\begin{itemize}
  \item \texttt{3.\_describe\_compound\_structure\_sorts(e) \texttt{Descriptor}}
\end{itemize}

\begin{verbatim}
let \{"E_1, \ldots, E_m"\} = analyse\_compound\_structure\_sorts(e) in

Narration:
\begin{itemize}
  \item \texttt{type} \texttt{\[s\]} \ldots narrative text on sorts ...
  \item \texttt{value} \texttt{\[o\]} \ldots narrative text on sort observers ...
  \item \texttt{proof obligation} \texttt{\[p\]} \ldots narrative text on proof obligations ...
\end{itemize}

Formalisation:
\begin{itemize}
  \item \texttt{type name, type of, is\_}:
\end{itemize}

The definition of \texttt{obs\_E}\_i implicitly implies the definition of

\begin{itemize}
  \item \texttt{obs\_E}_i(e) \equiv E_i \supset \text{type\_name}(e)_i \equiv E_i \land \text{type\_of}(e)_i \equiv E_i \land \text{is\_E}_i(e)_i
\end{itemize}

Elaboration 4 Type, Values and Type Names: Note the use of quotes above Please observe that when we write \texttt{obs\_E} then \texttt{obs\_E} is the name of a function. The \texttt{E}, when juxtaposed to \texttt{obs\_} is now a name.

Analysis Function Prompt 11 \texttt{type\_name, type\_of, is\_:}

\begin{itemize}
  \item \texttt{obs\_E}_i(e) \equiv E_i \supset \text{type\_name}(e)_i \equiv E_i \land \text{type\_of}(e)_i \equiv E_i \land \text{is\_E}_i(e)_i
\end{itemize}

3.18.3 Describing Set Structures

Recall, from Sect. 3.10.3, that there are two kinds of structural endurants: compound and set. We now treat the set structures.
Elaboration 5 Type, Values and Type Names: Note the use of quotes above. Please observe that when we write $\mathrm{obs}_P$ then $\mathrm{obs}_P$ is the name of a function. The $P$, when juxtaposed to $\mathrm{obs}_P$ is now a name.

Analysis Function Prompt 12 type_name, type_of, is_:

The definition of $\mathrm{obs}_P$ implicitly implies the definition of

$$\mathrm{obs}_P(e) \equiv \text{type_name}(p) \equiv P \land \text{type_of}(p) \equiv P \land \text{is}_P(p)$$

Modelling Choice 1 Endurants: For composite endurants and structures the analyser cum describer chooses for some model of a domain, one subset of the endurants forming the composite or structure, for another model of supposedly “the same” domain another subset.

Method Step 8 Put New Endurants on Hold:

Having described composite or structure endurant $e$ into a set of endurants of type names $\{E_1, E_2, \ldots, E_m\}$, these latter endurant names if new, are now remembered, that is, put on hold say on a notice board, for later analysis & description.

In preparation for Examples 38–39 on the next page and 41 on Page 72 we show Fig. 3.4 on the next page.

Example 38 46

117 There is the universe of discourse, $\text{UoD}$. 118 a road net, RN, and
It is structured into 119 a fleet of vehicles, FV.
Fig. 3.4. A Road Transport System: Structures and Parts

Both are structures.

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>117 UoD axiom ( \forall uod:UoD \cdot is_structure(uod) )</td>
<td>118 obs_RN: UoD ( \rightarrow ) RN</td>
</tr>
<tr>
<td>118 RN axiom ( \forall rn:RN \cdot is_structure(rn) )</td>
<td>119 obs_FV: UoD ( \rightarrow ) FV</td>
</tr>
<tr>
<td>119 FV axiom ( \forall fv:FV \cdot is_structure(fv) )</td>
<td></td>
</tr>
</tbody>
</table>

A Road Transport Domain, II: Structure

Example 39

120 The road net consists of
   a a structure, \( SH \), of hubs and
   b a structure, \( SL \), of links.

121 The fleet of vehicles consists of
   a a structure, \( SBC \), of bus companies, and
   b a structure, \( PA \), a pool of automobiles.

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>120a SH axiom ( \forall sh:SH \cdot is_structure(sh) )</td>
<td>120a obs_SH: RN ( \rightarrow ) SH</td>
</tr>
<tr>
<td>120b SL axiom ( \forall sl:SL \cdot is_structure(sl) )</td>
<td>120b obs_SL: RN ( \rightarrow ) SL</td>
</tr>
<tr>
<td>121a SBC axiom ( \forall sbc:SBC \cdot is_structure(bc) )</td>
<td>121a obs_BC: FV ( \rightarrow ) BC</td>
</tr>
<tr>
<td>121b PA axiom ( \forall pa:PA \cdot is_structure(pa) )</td>
<td>121b obs_PA: FV ( \rightarrow ) PA</td>
</tr>
</tbody>
</table>

3.18.4 Describing Conjoins

We refer to Sects. 3.13.4, 3.13.5 and 3.13.6 for our initial treatment of conjoins.

Example 38 on the facing page’s Narration is not representative of what it should be. Here is a more reasonable narration:

- A road net is a set of hubs (road intersections) and links such that links are connected to adjacent hubs, and such that connected links and hubs form roads and where a road is a thoroughfare, route, or way on land between two places that has been paved or otherwise improved to allow travel by foot or some form of conveyance, including a motor vehicle, cart, bicycle, or horse [Wikipedia]

Et cetera for fleet of vehicles.

We bring this clarification here, once, and allow ourselves, with the reader’s permission, to narrate only very steno-graphically.
3.18.4.1 Describe Material-Parts Sorts

Domain Description Prompt 5 \texttt{describe\_material\_parts\_sorts}: The \textit{domain description prompt}:

\begin{itemize}
  \item \texttt{describe\_material\_parts\_sorts(e)}
\end{itemize}

yields the \textit{material-parts conjoin sorts and conjoin sort observers} domain description text according to the following schema:

\begin{center}
5. \texttt{describe\_material\_parts\_sorts(e)}
\end{center}

\begin{Verbatim}
let (_, \texttt{M}) = analyse\_material\_parts\_material(e),
  (_, \texttt{P}) = analyse\_material\_parts\_parts(e) in
\end{Verbatim}

Narration:
- \[s\] ... narrative text on conjoin sorts ...
- \[o\] ... narrative text on conjoin sort observers ...

Formalisation:
- \texttt{type}
  \[s\] M, P
- \texttt{value}
  \[o\] \texttt{obs\_M}: E \rightarrow M
  \[o\] \texttt{obs\_P}: E \rightarrow P

\textbf{Analysis Function Prompt 13} \texttt{type\_name, type\_of, is\_}:

The definitions of \texttt{obs\_M}: E \rightarrow M and \texttt{obs\_P}: E \rightarrow P implicitly imply the definition of

\begin{itemize}
  \item \texttt{obs\_M}(e) = m \circ \textit{type\_name}(m) \equiv \texttt{M}
  \item \texttt{obs\_P}(e) = p \circ \textit{type\_of}(p) \equiv \texttt{P}
  \item \texttt{obs\_M}(e) = m \circ \textit{type\_name}(m) \equiv M
  \item \texttt{obs\_P}(e) = p \circ \textit{type\_of}(p) \equiv P
\end{itemize}

\textbf{Modelling Choice 2 Material-Parts}: As for composite and structure endurants the analyser cum describer chooses for some model of a domain, one subset of the parts forming the conjoin, for another model of supposedly “the same” domain another subset.
3.18.4.2 Describe Part-Materials Sorts

Domain Description Prompt 6 `describe_part_materials_sorts`: The **domain description prompt**:  
- `describe_part_materials_sorts(e)`  
  yields the **conjoin sorts and conjoin sort observers** domain description text according to the following schema:  

<table>
<thead>
<tr>
<th>6. describe_part_materials_sorts(e) Descriptor</th>
</tr>
</thead>
<tbody>
<tr>
<td>let (__,(M1,...,Mm)) = analyse_part_materials_materials(e) in</td>
</tr>
<tr>
<td><strong>Narration:</strong></td>
</tr>
<tr>
<td>[s] ... narrative text on conjoin sorts ...</td>
</tr>
<tr>
<td>[o] ... narrative text on conjoin sort observers ...</td>
</tr>
<tr>
<td><strong>Formalisation:</strong></td>
</tr>
<tr>
<td>type</td>
</tr>
<tr>
<td>[s] M1, ..., Mm</td>
</tr>
<tr>
<td>value</td>
</tr>
<tr>
<td>[o] obs,M1: E → M1, ..., obs,Mm: E → Mm</td>
</tr>
</tbody>
</table>

We shall mostly associate more than one material with a special kind of conjoins: the so-called **treatment** conjoins, leaving all other conjoins to embody just one material.

**Analysis Function Prompt 14** `type_name, type_of, is_:`  
The definitions of `obs,Mi: E -\rightarrow Mi` implicitly imply the definition of  
- `obs,Mi(e) = mo \supset type_name(mi) \equiv \langle Mi \rangle`  
- `obs,Mi(e) = mi \supset type_of(mi) \equiv Mi`

**Modelling Choice 3** **Part-Materials**: As for composites, structures and, now, in general for conjoins the analyser cum describer chooses for some model of a domain, one subset of the materials forming the conjoin, for another model of supposedly “the same” domain another subset.

**Pipeline Parts and Material**

**Example 40** We refer to Appendix Sect. A.1.
3.18.4.3 Describe Part-Parts Sorts

Domain Description Prompt 7 describe_part_parts_sorts: The domain description prompt:

- describe_part_parts_sorts(e)

yields the conjoin sorts and conjoin sort observers domain description text according to the following schema:

```
let (_, P) = analyse_part_parts(e) in

Narration:
[ ] ... narrative text on conjoin sorts ...
[ ] ... narrative text on conjoin sort observers ...

Formalisation:

<table>
<thead>
<tr>
<th>type</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td></td>
</tr>
<tr>
<td>[0] obs_P: E → P</td>
<td></td>
</tr>
</tbody>
</table>

```

Analysis Function Prompt 15 type_name, type_of, is: 

The definitions of obs_P: E → P implicitly imply the definition of

- obs_P(e) = p ⊃ type_name(p) ≡ "P"
- obs_P(e) = p ⊃ type_of(p) ≡ P

Modelling Choice 4 Part-Parts: As for composites, structures and, now, in general for conjoins the analyser cum describer chooses for some model of a domain, one subset of the parts forming the conjoin, for another model of supposedly “the same” domain another subset.

Example 41

122 The structure of hubs is a set, sH, of atomic hubs, H.
123 The structure of links is a set, sL, of atomic links, L.
124 The structure of buses is a set, sBC, of composite bus companies, BC.
125 The composite bus companies, BC, are sets of buses, sB.
126 The structure of private automobiles is a set, sA, of atomic automobiles, A.

122 H, sH = H-set axiom ∀ h:H · is_atomic(h)
123 L, sL = L-set axiom ∀ l:L · is_atomic(l)
124 BC, sBC = BC-set axiom ∀ bc:BC · is_composite(bc)
125 B, sB = B-set axiom ∀ b:B · is_atomic(b)
126 A, sA = A-set axiom ∀ a:A · is_atomic(a)

value
122 obs_sH: SH → sH
123 obs_sL: SL → sL
124 obs_sBC: SBC → BCs
125 obs_sB: BCs → Bs
126 obs_sA: SA → sA

© Dines Bjørner 2020, Fredsvej 11, DK–2840 Holte, Denmark
Domain Science & Engineering July 6, 2020, 10:03
3.18.5 Describing Material

We refer to Sect. 3.12 on Page 54 for our initial treatment of ‘materials’. We suggest a domain analysis function: \( \text{analyse}\_\text{material} \) which, when “applied” to a material endurant, \( \text{is}\_\text{material}(e) \), observes that material’s value and type.

If \( \text{is}\_\text{material}(e) \) holds then we can apply the domain description prompt: \( \text{observe}\_\text{material}\_\text{sort}(e) \).

Domain Description Prompt 8 \( \text{describe}\_\text{material}\_\text{sort} \):
The domain description prompt:

\[ \text{describe}\_\text{material}\_\text{sort}(e) \]

yields the material sort and material sort observer domain description text according to the following schema whether the endurants \( e \) actually contains materials:

\[
\begin{array}{ll}
\text{let } & m \text{ = analyse}_\text{material}(e) \\
\text{Narration: } & s \text{ ... narrative text on material sort ...} \\
\text{Narration: } & o \text{ ... narrative text on material sort observer ...} \\
\text{Formalisation: } & \\
\text{type } & s \text{ M} \\
\text{value } & o \text{ obs}_\text{M}: E \rightarrow M \\
\end{array}
\]

Analysis Function Prompt 16 \( \text{type}_\text{name}, \text{type}_\text{of}, \text{is}_\text{.} \):
The definitions of \( \text{obs}_\text{M}: E \rightarrow M \) implicitly imply the definition of

\[ \begin{array}{l}
\text{obs}_\text{M}(e) = m \circ \text{type}_\text{name}(m) \\
\text{obs}_\text{M}(e) = m \circ \text{type}_\text{of}(m) \\
\end{array} \]

3.19 On Endurant Sorts

3.19.1 Derivation Chains

Let \( E \) be a composite sort or a structure. Let \( E_1, E_2, \ldots, E_m \) be the endurants “discovered” by means of \( \text{observe}\_\text{endurant}\_\text{sorts}(e) \) where \( e:E \). We say that \( E_1, E_2, \ldots, E_m \) are (immediately) \( \text{derived} \) from \( E \). If \( E_k \) is derived from \( E_j \) and \( E_j \) is derived from \( E_i \), then, by transitivity, \( E_k \) is \( \text{derived} \) from \( E_i \).

3.19.2 No Recursive Derivations:

We “mandate” that if \( E_k \) is derived from \( E_j \) then sort name \( E_j \) is different from sort name \( E_k \) and there can be no \( E_k \) derived from \( E_j \), that is, \( E_k \) cannot be derived from \( E_k \). That is, we do not “provide for” recursive domain sorts. It is not a question, actually, of allowing recursive domain sorts. It is, we claim to have observed, in very many \( \text{analysis & description} \) experiments, that there are no recursive domain sorts.

\[ 47 \text{ Some readers may object, but we insist! If trees are brought forward as an example of a recursively definable domain, then we argue: Yes, trees can be recursively defined, but it is not recursive. Trees can, as well, be defined as a variant of graphs, and you wouldn’t claim, would you, that graphs are recursive?} \]
3.19.3 Names of Part Sorts and Types

The **domain analysis & description** text prompts observe endurant sorts, as well as the below-defined observe part type, observe component sorts and observe material sorts – as well as the further below defined attribute names, observe material sorts, observe unique identifiers, observe mereology and observe attributes prompts introduced below – “yield” type names. That is, it is as if there is a reservoir of an indefinite-sized set of such names from which these names are selected, and once obtained are never again selected. There may be domains for which two distinct part sorts may be composed from identical part sorts. *In this case the domain analyser indicates so by prescribing a part sort already introduced.*

3.20 A Domain Discovery Process, I

In this and some following sections we shall clarify some aspects of the **domain analysis & description** method. A method principle is that of exhaustively analyse & describe all external qualities of the domain under scrutiny. A method technique implied here is that sketched in Sect. 4.8 on Page 116. The method tools are here all the analysis and description prompts covered so far.

In this initial chapter on **domain analysis & description** we have systematically covered, first, the analysis of external qualities of domain endurants, then the description of these. We have done so in a style which **analysed domains**, as it were, “top-down”; from overall domain universes of discourse; through entities, endurants, discrete and continuous (material) endurants; further “across” physical parts, structures and living species; the natural and artefactual parts of the physical parts; to finally conclude our external qualities analysis with the atomic, composite and conjoin parts. With the ontology of the external qualities of domain endurants “behind us” we then concluded the main sections of this chapter with the description of external domain qualities, that is, **Describer Schemas 1** on Page 41, 2 on Page 66, 3 on Page 67, 4 on Page 68, 5 on Page 70, 6 on Page 71, 7 on Page 72 and 8 on the preceding page. We can now gather all of this together with advice on a systematic process of performing the analysis & description process. Chapters 4 (Sect. 4.8 on Page 116) and 6 (Sect. 6.12 on Page 155) will likewise systematise the processes of discovering internal endurant qualities and perdurants, respectively.

3.20.1 A Domain Discovery Notice Board

Common to all the discovery processes is an idea of a notice board. A notice board, at any time in the development of a domain description, is a repository of the analysis and description process. We suggest to model the notice board in terms of three global variables. The new variable holds the parts yet to be described. The ans variable holds the sort name of parts that have so far been described, the gen variable holds the parts that have so far been described, and the txt variable holds the RSL-Text so far generated. We model the txt variable as a map from endurant identifier names to RSL-Text.

```
<table>
<thead>
<tr>
<th>A Domain Discovery Notice Board</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
</tr>
<tr>
<td>new := {uod}</td>
</tr>
<tr>
<td>asn := { «UoD»}</td>
</tr>
<tr>
<td>gen := {}</td>
</tr>
<tr>
<td>txt: RSL-Text ::= [ uid_UoD(uod) \mapsto {«type UoD»} ]</td>
</tr>
</tbody>
</table>
```

48 Sects. 4.8 on Page 116 and 6.12 on Page 155
3.20.2 An Endurant External Qualities Discovery Process

The `discover_sorts` pseudo program suggests a systematic way of proceeding through analysis, manifested by the `is_...` predicates, to (→) description.

Some comments are in order. The `e-set_un-e-set_e` expression yields a set of endurants that are either in `e-set_un`, or in `e-set_e`, or in both, but such that two endurants, `e_u` and `e_v` which are of the same endurants type, say `E`, and are in respective sets is only represented once in the result; that is, if they are type-wise the same, but value-wise different they will only be included once in the result. As this is the first time RSL-Text is put on the notice board we express this as:

- `txt := txt \cup [\text{name(v)} \mapsto \langle\text{RSL-Text}\rangle]`

Subsequent insertion of RSL-Text for internal quality descriptions and perdurants is then concatenated to the end of previously uploaded RSL-Text.

<table>
<thead>
<tr>
<th>An External Qualities Domain Analysis and Description Process</th>
</tr>
</thead>
</table>

```
value

discover_sorts: Unit \rightarrow Unit

discover_sorts() \equiv

  while new ≠ {} do let v \in new in ( new := new \setminus \{v\} \parallel gen := gen \cup \{v\} ) ;

  is_atomic(v) \rightarrow skip,

  is_composite(v) \rightarrow

    let \{E_1,...,E_n\} = analyse_endurant_sorts(v) in
    ( ans := ans \cup \{E_1,...,E_n\} \parallel new := new \parallel \{E_1,...,E_n\} )

    txt := txt \cup \{\text{name(v)} \mapsto \langle\text{describe_endurant_sorts(v)}\rangle\} \) end,

  is_structure(v) \rightarrow

    let \{E_1,...,E_n\} = analyse_endurant_sorts(v) in
    ( ans := ans \cup \{E_1,...,E_n\} \parallel v = new \parallel \{E_1,...,E_n\} ) end,

  is_conjoin(v) \rightarrow

    ( is_part_materials_conjoin(v) \rightarrow
      let \{e,m_1,...,m_n\} = analyse_part_materials(v) in
      new := new \parallel \{m_1,...,m_n\}
      txt := txt \cup \{\text{name(v)} \mapsto \langle\text{describe_part_materials_sorts(v)}\rangle\} end,

    is_material(v) \mapsto

      let m = analyse_material_part_material(v),
      \{p_1,...,p_n\} = analyse_material_part_material(v) in
      ( ans := ans \cup \{E_1,...,E_n\} \parallel new := new \parallel \{p_1,...,p_n\} )
      || txt := txt \cup \{\text{name(v)} \mapsto \langle\text{describe_material_part_materials_sorts(v)}\rangle\} ) end,

    is_part_parts_conjoin(v) \rightarrow

      let \{p_1,...,p_n\} = analyse_part_parts(v) in
      ( ans := ans \cup \{E_1,...,E_n\} \parallel new := new \parallel \{p_1,...,p_n\} )
      || txt := txt \cup \{\text{name(v)} \mapsto \langle\text{describe_part_parts_sorts(v)}\rangle\} end ) end

end
```

3.20.3 An Assumption

In the above `External Qualities Domain Analysis and Description Process` Schema we have conjectured that atomic parts have already had their type and observer function defined.

---

For structures we remove the structure endurants, here `v`, from `ans` as it has no “root” part to be further analysed and described. This marks – a major – difference between composite endurants and structure endurants.
3.21 Formal Concept Analysis

Domain analysis involves that of concept analysis. As soon as we have identified an entity for analysis, we have identified a concept. The entity is usually a spatio-temporal, i.e., a physical thing. Once we speak of it, it becomes a concept. Instead of examining just one entity the domain analyser shall examine many entities. Instead of describing one entity the domain describer shall describe a class of entities. Ganter & Wille’s [158] addresses this issue.

3.21.1 A Formalisation

This section is a transcription of Ganter & Wille’s [158] Formal Concept Analysis, Mathematical Foundations, the 1999 edition, Pages 17–18.

Some Notation: By \( E \) we shall understand the type of entities; by \( E \) we shall understand a phenomenon of type \( E \); by \( Q \) we shall understand the type of qualities; by \( Q \) we shall understand a quality of type \( Q \); by \( E \)-set we shall understand the type of sets of entities; by \( ES \) we shall understand a set of entities of type \( E \)-set; by \( Q \)-set we shall understand the type of sets of qualities; and by \( QS \) we shall understand a set of qualities of type \( Q \)-set.

Definition: 54 Formal Context: A formal context \( K := (ES, I, QS) \) consists of two sets; \( ES \) of entities and \( QS \) of qualities, and a relation \( I \) between \( E \) and \( Q \).

To express that \( E \) is in relation \( I \) to a Quality \( Q \) we write \( E \cdot I \cdot Q \), which we read as “entity \( E \) has quality \( Q \)”.

Example endurant entities are a specific vehicle, another specific vehicle, etcetera; a specific street segment (link), another street segment, etcetera; a specific road intersection (hub), another specific road intersection, etcetera; a monitor.

Example endurant entity qualities are (a vehicle) has mobility, (a vehicle) has velocity (\( \geq 0 \)), (a vehicle) has acceleration, etcetera; (a link) has length (\( > 0 \)), (a link) has location, (a link) has traffic state, etcetera.

Definition: 55 Qualities Common to a Set of Entities: For any subset, \( sES \subseteq ES \), of entities we can define \( DQ \) for “derive[d] set of qualities”:

\[
DQ(sES)(ES, I, QS) \equiv \{ Q | Q : Q \cdot E : E \in sES \wedge E \cdot I \cdot Q \}
\]

pre: \( sES \subseteq ES \)

The above expresses: “the set of qualities common to entities in \( sES \)”.

Definition: 56 Entities Common to a Set of Qualities: For any subset, \( sQS \subseteq QS \), of qualities we can define \( DE \) for “derive[d] set of entities”:

\[
DE(sQS)(ES, I, QS) \equiv \{ E | E : E \in sQS \wedge E \cdot I \cdot Q \},
\]

pre: \( sQS \subseteq QS \)

The above expresses: “the set of entities which have all qualities in \( sQS \)”.

Definition: 57 Formal Concept: A formal concept of a context \( K \) is a pair:

- \( (sQ, sE) \) where
- \( DQ(sE)(E, I, Q) = sQ \) and
- \( DE(sQ)(E, I, Q) = sE \).

\( sQ \) is called the intent of \( K \) and \( sE \) is called the extent of \( K \).
3.21.2 Types Are Formal Concepts

Now comes the “crunch”: In the TripTych domain analysis we strive to find formal concepts and, when we think we have found one, we assign a type (or a sort) and qualities to it!

3.21.3 Practicalities

There is a little problem. To search for all those entities of a domain which each have the same sets of qualities is not feasible. So we do a combination of two things: (i) we identify a small set of entities all having the same qualities and tentatively associate them with a type, and (ii) we identify certain nouns of our national language and if such a noun does indeed designate a set of entities all having the same set of qualities then we tentatively associate the noun with a type. Having thus, tentatively, identified a type we conjecture that type and search for counterexamples, that is, entities which refute the conjecture. This “process” of conjectures and refutations is iterated until some satisfaction is arrived at that the postulated type constitutes a reasonable conjecture.

3.21.4 Formal Concepts: A Wider Implication

The formal concepts of a domain form Galois Connections [158]. We gladly admit that this fact is one of the reasons why we emphasise formal concept analysis. At the same time we must admit that this paper does not do justice to this fact. We have experimented with the analysis & description of a number of domains, and have noticed such Galois connections, but it is, for us, too early to report on this. Thus we invite the reader to study this aspect of domain analysis.

3.22 Summary

This chapter’s main title was: DOMAINS – A Taxonomy. So, the taxonomy of a domain, such as we have studied it and such as we ordain one aspect of domain analysis & description, is about manifestly visible and tangible properties, that is, the external qualities. For that study & practice we have suggested a number of analysis & description prompts.

3.22.1 The Description Schemas

We have culminated this chapter with the description prompts: endurant describer 2 on Page 66, compound structures describer 3 on Page 67, set structures describer 4 on Page 68, material-parts describer 5 on Page 70, part-materials describer 6 on Page 71, part-parts describer 7 on Page 72 and material describer 8 on Page 73.

They all describe the, in our case RSL, domain description text to ‘produce’ when external quality analysing & describing a given endurant; but what about the description of those endurants revealed by the analysis & description of that given endurant?

The answer is simple. That is up to you! The domain analysis & description method primarily gives you the tools. But!

• A principle of the method could be to secure that all relevant, i.e., implied, endurants are analysed & described.
• A technique could be to, somehow, “set aside” all those endurants revealed by the analysis & description of any given endurant – with the proviso that no endurant, of type, for example, \( P \), is analysed & described more than once.

We refer to Sect. 3.20 on Page 74 for a suggested analysis & description technique (cum pseudo program expressed in pseudo RSL).
3.22.2 Modelling Choices

In this chapter we have put forward some advice on description choices: We refer to Modelling Choices 1 on Page 68, 2 on Page 70, 3 on Page 71 and 4 on Page 72. The analysis predicates and functions are merely aids. They do not effect descriptions, but descriptions are based on the result of inquiries based on deployment of these predicates and functions. Real decisions are made when effecting a description function. So the rôle of these modelling choice paragraphs is to alert the describer to make judicious choices.

3.22.3 Method Principles, Techniques and Tools

Recall that by a method we shall understand a set of principles for selecting and applying a set of techniques using a set of tools in order to construct an artefact.

3.22.3.1 Principles of External Qualities

In this chapter we have illustrated the use of the following principles:

Divide and Conquer: We claim that the divide principle is applied in establishing the ontology: in distinguishing between describables and non-describables, in distinguishing between endurants and perdurants, and in otherwise suggesting the taxonomy as illustrated in Fig. 3.1 on Page 44. We claim that a guiding principle in this “division” has been Kai Sørlander’s Philosophy. And we claim that this “division” has helped and will help “conquer” the complexity of issues as they continue to unfold in the next chapters.

Abstraction: This principle is applied in simply focusing on abstract names for endurant sorts, disregarding any further meaning of these names – meanings that will be “revealed” as we go along in analysing as describing, in the next chapters, first unique identifiers, mereologies and attributes, then the elements of perdurants.

Narration & Formalisation: This principle is applied in developing and presenting the domain endurant descriptions which are always, as shown in both the description schemas and in those examples which do present formalisations, in that they also show narratives.

3.22.3.2 Techniques of External Qualities

In this chapter we have illustrated the use of the following techniques:

Model-oriented Specification: Although we say model-oriented, there really are three aspects to our formal specifications: the use of discrete mathematics – so far logic, sets, Cartesians; the use of RSL’s specification/programming-like constructs: type definitions, function signatures, etc.; and the use of abstract sorts – as “inspired” from algebraic specifications.

Formal Concept Analysis: This technique, whose mathematical foundation was outlined in Sect. 3.21, involves “top-down” analysis, from most abstract concepts towards less and less abstract concepts, versus “bottom-up” analysis i.e., the “other way around”. We refer to Sect. 3.14.

3.22.3.3 Tools of External Qualities

The main tools are the English language, used in narrative descriptions, the RAISE Specification Language RSL, used in formal descriptions, and the analysis and description prompts – reviewed below – and as used by the domain analyser & describer, but a use that may not necessarily be explicitly recorded, as their “existence” are to mainly serve as aide-mémoire.

50 – thus accounting for our use of the term ‘model-oriented’
3.22.3.3.1 Tools: Review of the Analysis Calculus

In this chapter we have introduced a number of external qualities analysis prompts. Let $\pi$ designate a *phenomena*. The following are some of the external qualities analysis prompts.

- **is_entity($\pi$).**
  - If $\text{is}\_\text{entity}(\pi)$ then either it
    - $\text{is}\_\text{endurant}(\pi)$ or
    - $\text{is}\_\text{perdurant}(\pi)$.
  - If an entity, $e$, $\text{is}\_\text{endurant}(e)$, then either it
    - $\text{is}\_\text{discrete}(e)$ or
    - $\text{is}\_\text{material}(e)$.
  - If an endurant $\text{is}\_\text{discrete}(e)$, then either it
    - $\text{is}\_\text{physical}\_\text{part}(e)$ or
    - $\text{is}\_\text{structure}(e)$ or
    - $\text{is}\_\text{living}\_\text{species}(e)$.
  - If an endurant $\text{is}\_\text{physical}\_\text{part}(e)$ then either it
    - $\text{is}\_\text{natural}\_\text{part}(e)$ or
    - $\text{is}\_\text{artefact}(e)$.
  - If an endurant $\text{is}\_\text{structure}(e)$ then either it
    - $\text{is}\_\text{compound}\_\text{structure}(e)$ or
    - $\text{is}\_\text{set}\_\text{structure}(e)$ or
    - $\text{is}\_\text{living}\_\text{species}(e)$.
  - If an endurant $\text{is}\_\text{natural}\_\text{part}(e)$ then either it
    - $\text{is}\_\text{plant}(e)$ or
    - $\text{is}\_\text{animal}(e)$.
  - Some animals satisfy
    - $\text{is}\_\text{human}(e)$. 15 Pg. 54
  - If an endurant $\text{is}\_\text{natural}\_\text{part}(e)$ then either it
    - $\text{is}\_\text{atomic}(e)$ or
    - $\text{is}\_\text{composite}(e)$.
  - If a discrete endurant $\text{is}\_\text{artefact}(e)$ then either it
    - $\text{is}\_\text{atomic}(e)$ or
    - $\text{is}\_\text{composite}(e)$ or
    - $\text{is}\_\text{conjoin}(e)$.
  - If an artefact $\text{is}\_\text{conjoin}(e)$ then either it
    - $\text{is}\_\text{part}\_\text{materials}\_\text{conjoin}(e)$ or
    - $\text{is}\_\text{material}\_\text{parts}\_\text{conjoin}(e)$ or
    - $\text{is}\_\text{part}\_\text{parts}\_\text{conjoin}(e)$.

3.22.3.3.2 Tools: Review of the Description Calculus

And in this chapter we have introduced a number of external qualities description prompts. The following are some of the external qualities description prompts.

- **name_and_sketch_universe_of_discourse.**
  - **describe_endurant_sorts(e).** 1 Pg. 41
  - **describe_compound_structure_sorts(e).**
  - **describe_set_structure_sorts(e).**
  - **describe_material_parts_sorts(e).**
  - **describe_part_materials_sorts(e).**
  - **describe_part_parts_sorts(e).**
  - **describe_material_sort(e).**
3.22.4 How Much or How Little Do We Analyse and Describe?

How many of a domain’s external qualities do we analyse and describe? There are two kinds of answers to this question. **An Engineering Answer:** This kind of answer may be relevant for the case of a full scale software development – where a domain engineering phase is followed by a requirements engineering phase which is then followed by a software design phase. We may then try to capture just what we think we need for that subsequent requirements capture, its analysis and prescription. Or, to “guard against unforeseen eventualities”, a little more! Reading engineering domain analysis & description case studies helps. So do experience! **A Scientific Answer:** Or we try to capture “all”! Now that is clearly not possible, at least not “in one fell swoop”\(^{51}\)! So how do we go about it, as domain scientists cum engineers? We do it “domain-area-by-domain-area”. Sort of, for example like this: First what is thought of as a core domain is analysed & described. Then some additional aspects, i.e., entities, are included in a next analysis & description – leaving out, typically, some initially analysed & described entities. and so on. Just like, for example, physicists, analyse & describe natural world phenomena.

We shall have more to say about what to include and what to exclude in the next chapters.

3.23 Bibliographical Notes

We refer to [70, Sect. 5.3] for a thorough, 2016–2017, five page review of types in formal specification and programming languages.

3.24 Exercise Problems


3.24.1 Research Problems

**Exercise 1** A Research Challenge. Reformulate Composites as Conjoins: In this chapter we have treated artefactual composites apart from conjoins. But, really, are these artefacts not also conjoins? Reformulate the appropriate text to reflect this “change of ontology”!

**Exercise 2** A Research Challenge. Symmetry of Part-Parts Conjoins: Sets versus Composites: In this chapter we have suggested Material-Parts, Part-Materials and Part-Parts conjoins. The ‘plural’ s in material-parts means that we allow a set, more precisely, an indefinite number of parts; the plural s in part-materials means that we expect either a single or a Cartesian of a definite number of materials, expressed as a Cartesian; and the ‘plural’ s in part-parts means that we allow sets of parts.

[Q1] Consider a Part-Cartesian-Parts conjoin, almost like the Part-Parts conjoin but with a definite number of parts of possibly distinct sorts. How is that possibility different from the suggestion of research problem 1 above?

[Q2] Could one contemplate a variant Part-Materials variant where the s indicates that we now expect an in definite number of materials?

[Q3] Discuss those possibilities, [a–b], and reformulate ontology accordingly.

3.24.2 A Student Exercise

**Exercise 3** An MSc Student Exercise. Document System Parts: A document system consists of persons and documents. To anticipate exercise 29 on Page 160 we characterise, so that the reader can get at what we mean by documents, these as subject to the following operations:

\(^{51}\) To do something in one fell swoop is to do it suddenly or in a single, swift action.
[a] creation: before there might have been a number of unrelated documents – now there is a [new] document, with some text created and written by a person; [b] editing: before there was a document – now there is a document with text and editing being done by a person; [c] reading: before there was a document – now there is “the same” document, only now it has been read by a person; [d] copying: before there was a document – now there is “the same” document, only now it has been copied by a person – and there is a copy (of the former, still existing, separate document) identifying that (former) document and with all the “contents” of the “original” of which it is a copy – the ‘copy’ creator is also identified; [e] shredding: before there was a document – now that document no longer “exists” – but otherwise all other documents remain unchanged!

[Q1] You are to narrate and formalise the parts of the document system.

[Q2] Are shredded documents to be a part of the system?

This exercise is continued in Exercises 16 on Page 120, 17 on Page 120, 18 on Page 120 and 29 on Page 160.

3.24.3 Term Projects

In a textbook as this we cannot primarily rely on simple 10 line problems. It should be clear to the reader: lecturer and student, that exercise problems must be more-or-less comprehensive; they must encompass a reasonably well-delineated domain. We now list a number of such potential problem domains:

1. the consumer, retailer, wholesaler, etc., merchandise market;
2. financial service industry;
3. container line industry – with the (possibly overlapping) subdomain:
   a. container terminal ports,
   b. container stowage, and
   c. container logistics;
4. railways systems;
5. waste disposal systems,

We suggest that the lecturer, who is using this primer for a dedicated series of lectures on domain analysis & description,

• “divide” the class students into one or more groups of preferably 4–6 students each.

• that each group be assigned a distinct domain.

52 We refer to a number of experimental domain analysis & description reports:

• 2019: Container Terminal Ports, ECNU, Shanghai, China URL: imd.dtu.dk/dibj2018/yangshan/-maersk-pa.pdf
• 2018: Documents, TongJi Univ., Shanghai, China URL: imd.dtu.dk/dibj2017/docs/docs.pdf
• 2013: Road Transport, Techn. Univ. of Denmark URL: imd.dtu.dk/dibj/road-p.pdf
• 2012: Credit Cards, Univ. of Uppsala, Sweden URL: imd.dtu.dk/dibj/2016/credit/access.pdf
• 2010: Web-based Transaction Processing, Techn. Univ. of Vienna, Austria URL: imd.dtu.dk/dibj/ wfdftp.pdf
• 2009: Pipelines, Techn. Univ. of Graz, Austria URL: imd.dtu.dk/dibj/pipe-p.pdf
• and that, from week-to-week they discuss and write down their analysis and description (narratives and formalisations) of that domain, in phases corresponding to the ‘Exercise Problem’ Sects. 3.24.3 (the next section), and forthcoming sections: 4.12.3, 6.14.3, 7.11.2 and 8.9.2.

• For teachers and individual students the publisher provides access to “large scale” examples covering several of the exercise domains that we have listed.

We shall briefly illustrate some external quality aspects of these domains,

• in a first week of study, completely unstructured – since, you have not yet learned the full contents of this chapter, “rambling on”; to be followed,

• in a second week of study, once you have learned the “stuff” of this chapter, more structured, and according to the concepts of this chapter.

Exercise 4 An MSc Student Exercise. The Consumer Market, External Qualities: You are to analyse and describe the external qualities of ‘the market’ domain of artefactual entities including consumers, retailers, wholesalers, possibly importers/exporters, and producers of merchandise aimed at ordinary consumers.

Exercise 5 An MSc Student Exercise. Financial Service Industry, External Qualities: You are to analyse and describe the external qualities of a financial service industry domain of artefactual entities including banks, insurance companies, mortgage institutions, brokers and stock exchanges. Of what discrete endurants consists the banks, insurance companies, mortgage institutions, brokers and stock exchanges?

Exercise 6 An MSc Student Exercise. Container Line Industry, External Qualities: You are to analyse and describe the external qualities of a container line industry domain of artefactual entities including containers, container vessels, container terminal ports, trucks (transporting containers between customers and terminal ports), and the container line management. Of what discrete endurants consists these and related discrete endurants?

Exercise 7 An MSc Student Exercise. Railway Systems, External Qualities: You are to analyse and describe the external qualities of a railway domain of artefactual entities including trains and railway nets. Of what discrete endurants consists the trains, and of what discrete endurants consists the railway nets?

Exercise 8 A PhD Student Problem. Part-Material Conjoins: Canals, External Qualities: We refer to Example 28 on Page 58. You are to analyse and describe the external qualities of a canal system of artefactual entities including locks, straight (if curved) stretches of canals, and canal forks and joins (diverting, respectively collecting) water flows.

Exercise 9 A PhD Student Problem. Part-Materials Conjoins: Rum Production, External Qualities: We refer to Example 15 on Page 50. You are to analyse and describe the external qualities of a rum production system of artefactual entities including sugar cane fields, transport links from fields to sugar cane chopping facilities, such facilities, from these to the rum distillery, rum distilleries with their pot- or column stills and other production means, ware houses, an so forth.

Exercise 10 A PhD Student Problem. Part-Materials Conjoins: Waste Management, External Qualities: We refer to Example 29 on Page 58. You are to analyse and describe a waste management systems domain of artefactual entities, say focusing on just the (a) waste conveyors (whether ‘belts’ or ‘pipe’) and (b) waste processors: (a) conveyor belts or pipes, their “merging” and “diversion” (joins and forks), their initial sources and ultimate sinks, whether pumps (as for pipe) or no such things (as for mechanically moving belts that either move goods upwards, horisontally, or downwards, etc.); (b) industrial,
sewage, agricultural product, leachate\textsuperscript{53}, or other, biological, etc., treatment. Note that conveyor nets are directional and have no cycles.

These exercise problems are continued in Sects. 4.12.3 on Page 121, 6.14.3 on Page 161, 7.11.2 on Page 193 and 8.9.2 on Page 241.

\textsuperscript{53} A leachate is any liquid that, in the course of passing through matter, extracts soluble or suspended solids, or any other component of the material through which it has passed.
In this chapter we introduce the concept of internal qualities of endurants, and cover the analysis and description of unique identifiers, mereologies and attributes of endurants. There is yet another, interrelating internal quality: intentionality, “something” that expresses intention, design idea, purpose of artefacts – well, some would say, also natural endurants.

External qualities of endurants of a manifest domain are, in a simplifying sense, those we can see and touch. They so to speak, take form.

Internal qualities of endurants of a manifest domain are, in a less simplifying sense, those which we may not be able to see or “feel” when touching an endurant, but they can, as we now ‘mandate’ them, be reasoned about, as for unique identifiers and mereologies, or be measured by some physical/chemical means, or be “spoken of” by intentional deduction, and be reasoned about, as we do when we attribute properties to endurants.

As it turns out², to analyse and describe mereology we need first analyse and describe unique identifiers; and to analyse and describe attributes we need first analyse and describe mereologies. Hence:

Method Step 9  Sequential Analysis & Description of Internal Qualities:

We advice that the domain analyser & describer first analyse & describe unique identification of all endurant sorts; then analyse & describe mereologies of all endurant sorts; finally analyse & describe attributes of all endurant sorts.

In this monograph we shall not suggest the modelling of unique identifiers and mereology of materials. We shall comment on that in appropriate sections.

4.1 Overview of this Chapter

- Section 4.2 covers the crucial notion of unique identification of endurants;
- Sect. 4.3 the likewise important notion of mereology – relations between parts;
- Sect. 4.4 covers the notion of attributes, that, which in a sense, gives “flesh & blood” to endurants; and
- Sect. 4.5 covers the novel notion, in computing, that of “intentional pull”.
- Finally Sect. 4.8 follows up on the domain discovery process of Sect. 3.20.

Other sections provide elucidation or summary observations.

4.2 Unique Identifiers

The concept of parts having unique identifiability, that is, that two parts, if they are the same, have the same unique identifier, and if they are not the same, then they have distinct identifiers, that concept is fundamental to our being able to analyse and describe internal qualities of endurants. So we are left with the issue of “sameness”!

¹ The ‘Statics’ refer back to ‘DOMAINS’ – not to ‘Ontology’!
² You, the first time reader cannot know this, i.e., the “turns out”. Once we have developed and presented the material of this chapter, then you can see it; clearly!
4.2.1 On Uniqueness of Endurants

We therefore introduce the notion of unique identification of part endurants. We assume (i) that all part endurants, \( e \), of any domain \( E \), have unique identifiers, (ii) that unique identifiers (of part endurants \( e:E \)) are abstract values (of the unique identifier sort \( UI \) of part endurants \( e:E \)), (iii) such that distinct part endurant sorts, \( E_i \) and \( E_j \), have distinctly named unique identifier sorts, say \( UI_i \) and \( UI_j \), and (iv) that all \( ui_i:UI_i \) and \( ui_j:UI_j \) are distinct.

**Representation of Unique Identifiers:** Unique identifiers are abstractions. When we endow two endurants (say of the same sort) distinct unique identifiers then we are simply saying that these two endurants are distinct. We are not assuming anything about how these identifiers otherwise come about.

**Identifiability of Endurants:** From a philosophical point of view, and with basis in Kai Sørlander’s Philosophy, cf. Paragraph *Identity, Difference and Relations* (Page 14), one can rationally argue that there are many endurants, and that they are unique, and hence uniquely identifiable. From an empirical point of view, and since one may eventually have a software development in mind, we may wonder how unique identifiability can be accommodated.

Unique identifiability for discrete endurants even though they may be mobile, is straightforward: one can think of of many ways of ascribing a unique identifier to any part; discrete endurants do not “morph”\(^3\). Hence one can think of many such unique identification schemas.

Unique identifiability for materials may seem a bit more tricky. For this monograph we shall not suggest to endow materials with unique identification. We have simply not experimented with such part-materials and material-parts domains – not enough – to suggest so.

4.2.2 Uniqueness Modelling Tools

The analysis method offers an observer function \( uid_E \) which when applied to part endurants, \( e \), yields the unique identifier, \( ui:E \), of \( e \).

**Domain Description Prompt 9** `describe_unique_identifier`: We can therefore apply the domain description prompt:

- `describe_unique_identifier`

  to endurants \( e:E \) resulting in the analyser writing down the unique identifier type and observer domain description text according to the following schema:

```
• Narration:
  [s] ... narrative text on unique identifier sort UI ...
  [u] ... narrative text on unique identifier observer uid_E ...
  [a] ... axiom on uniqueness of unique identifiers ...

Formalisation:
  type
  [s] UI
  value
  [u] uid_E: E \( \rightarrow \) UI
```

`is_part(e)` is a prerequisite for `describe_unique_identifier(e)`.

The unique identifier type name, \( UI \) above, chosen, of course, by the domain analyser cum describer, usually properly embodies the type name, \( E \), of the endurant being analysed and mereology-described. Thus a part of type-name \( E \) might be given the mereology type name \( EI \). Generally we shall refer to these names by \( UI \).

\(^3\) This restriction is not necessary, but, for the time, we can assume that it is.

\(^4\) – from a state of being solid, but in various “shapes”, via states of melting, to states of vapour
4.2 Unique Identifiers

Analysis Function Prompt 17 type name, type of, is_:

Given description schema 9 we have, so-to-speak, “in-reverse” that
\[ \forall e : E \cdot \text{uid}_E(e) = \text{ui} \Rightarrow \text{type of} (\text{ui}) = \text{UI} \land \text{type name} (\text{ui}) = \text{❝} \text{UI} \text{❞} \land \text{is} \text{UI} (\text{ui}) \]

<table>
<thead>
<tr>
<th>Unique Identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 42</td>
</tr>
</tbody>
</table>

- We assign unique identifiers to all parts.
- By a road identifier we shall mean a link or a hub identifier.
- By a vehicle identifier we shall mean a bus or an automobile identifier.
- Unique identifiers uniquely identify all parts.
  - All hubs have distinct [unique] identifiers.
  - All links have distinct identifiers.
  - All bus companies have distinct identifiers.
  - All automobiles have distinct identifiers.
  - All parts have distinct identifiers.

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>H/UI</td>
<td>uid_H: H \rightarrow H/UI</td>
</tr>
<tr>
<td>L/UI</td>
<td>uid_L: H \rightarrow L/UI</td>
</tr>
<tr>
<td>BC/UI</td>
<td>uid_BC: H \rightarrow BC/UI</td>
</tr>
<tr>
<td>B/UI</td>
<td>uid_B: H \rightarrow B/UI</td>
</tr>
<tr>
<td>A/UI</td>
<td>uid_A: H \rightarrow A/UI</td>
</tr>
</tbody>
</table>

4.2.3 All Unique Identifiers of a Domain

Given a universe of discourse we can calculate the set of the unique identifiers of all its parts.

<table>
<thead>
<tr>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculate_all_unique_identifiers: UoD \rightarrow UI-set</td>
</tr>
<tr>
<td>calculate_all_unique_identifiers(uod) \equiv</td>
</tr>
<tr>
<td>let parts = calc_parts({uod}){} in</td>
</tr>
<tr>
<td>{ uid_E(e) \mid e : E \land e \in parts } end</td>
</tr>
</tbody>
</table>

Road Transport: Unique Identifier Auxiliary Functions

Example 43 Extract Parts from Their Unique Identifiers: .............................................

- From the unique identifier of a part we can retrieve, \( \rho \), the part having that identifier.

<table>
<thead>
<tr>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>P = H \mid L \mid BC \mid B \mid A</td>
</tr>
</tbody>
</table>

4.2.4 Unique Identifier Constants

Given a domain which do not “grow” or “shrink” in its number of observable endurants we can speak of the constancy of their sets of unique identifiers.
Uniqueness of Endurant Identifiers

All Parts of a Domain have Unique Identifiers

A Domain Law 1 All Parts of a Domain have Unique Identifiers:

- All parts of a described domain have unique identifiers.
Uniqueness of Road Net Identifiers

Example 45  We must express the following axioms:

144  All hub identifiers are distinct.
145  All link identifiers are distinct.
146  All bus company identifiers are distinct.
147  All bus identifiers are distinct.
148  All private automobile identifiers are distinct.
149  All part identifiers are distinct.

axiom
144  \( \text{card } h_s = \text{card } h_{ui}s \)
145  \( \text{card } l_s = \text{card } l_{ui}s \)
146  \( \text{card } bcs = \text{card } bc_{ui}s \)
147  \( \text{card } bs = \text{card } b_{ui}s \)
148  \( \text{card } as = \text{card } a_{ui}s \)
149  \( \text{card} \{h_{ui}s \cup l_{ui}s \cup bc_{ui}s \cup b_{ui}s \cup a_{ui}s\} = \text{card } h_{ui}s + \text{card } l_{ui}s + \text{card } bc_{ui}s + \text{card } b_{ui}s + \text{card } a_{ui}s \) ■

We ascribe, in principle, unique identifiers to all endurants whether natural or artefactual. We find, from our many experiments, cf. the Universes of Discourse example, Page 40, that we really focus on those domain entities which are artefactual endurants and their behavioural “counterparts”.

Pipeline Unique Identifiers

Example 46  We refer to Appendix Sect. A.2.

4.3  Mereology

We refer to introductory section Sect. 2.8.2 on mereology as a philosophical–logic subject and Appendix Sect. B for closing material on mereology. We shall not endow materials with mereologies. We shall comment on this in Sect. 4.3.5 on Page 92.

Definition: 58  Mereology: Mereology is the study and knowledge of parts and part relations ■

Mereology, as a logical/philosophical discipline, can perhaps best be attributed to the Polish mathematician/logician Stanisław Leśniewski [115, 61].

4.3.1  Endurant Relations

Which are the relations that can be relevant for “endurant-hood”? There are basically two relations: (i) physical ones, and (ii) conceptual ones.

(i) Physically two or more endurants may be topologically either adjacent to one another, like rails of a line, or within an endurant, like links and hubs of a road net, or an atomic part is conjoined to one or more materials, or a material is conjoined to one or more parts. The latter two could also be considered conceptual “adjacencies”.

(ii) Conceptually some parts, like automobiles, “belong” to an embedding endurant, like to an automobile club, or are registered in the local department of vehicles, or are ‘intended’ to drive on roads.
4.3.2 Mereology Modelling Tools

When the domain analyser decides that some endurants are related in a specifically enunciated mereology, the analyser has to decide on suitable mereology types and mereology observers (i.e., endurant relations).

We may, for illustration, define a mereology type of an endurant $e : E$ as a triplet type expression over set of unique [endurant] identifiers.

There is the identification of all those endurant sorts $E_{i_1}, E_{i_2}, ..., E_{i_m}$ where at least one of whose properties "is of interest" to parts $e : E$.

There is the identification of all those sorts $E_{i_01}, E_{i_02}, ..., E_{i_0n}$ where at least one of whose properties "is of interest" to endurants $e : E$ and vice-versa.

There is the identification of all those endurant sorts $E_{o_1}, E_{o_2}, ..., E_{o_o}$ for whom properties of $e : E$ "is of interest" to endurants of sorts $E_{i_01}, E_{i_02}, ..., E_{i_0n}$.

The mereology triplet sets of unique identifiers are disjoint and are all unique identifiers of the universe of discourse.

The triplet mereology is just a suggestion. As it is formulated here we mean the three ‘sets’ to be disjoint. Other forms of expressing a mereology should be considered for the particular domain and for the particular endurants of that domain. We leave out further characterisation of the seemingly vague notion "is of interest".

type
151 $iEI = iEI_1 | iEI_2 | ... | iEI_m$
152 $ioEI = ioEI_1 | ioEI_2 | ... | ioEI_n$
153 $oEI = oEI_1 | oEI_2 | ... | oEI_o$
150 $MT = iEI-set \times ioEI-set \times oEI-set$

axiom
154 $\forall (iset, ioset, oset) : MT \cdot$
154 $\text{card} \ iset + \text{card} \ ioset + \text{card} \ oset = \text{card} \cup \{iset, ioset, oset\}$
154 $\cup \{iset, ioset, oset\} \subseteq \text{calc all unique identifiers(uod)}$

Domain Description Prompt 10 describe_mereology(e): If has_mereology(p) holds for parts $p$ of type $P$, then the analyser can apply the domain description prompt:

• describe_mereology
to parts of that type and write down the mereology types and observer domain description text according to the following schema:

<table>
<thead>
<tr>
<th>10. describe_mereology(e) Observer</th>
</tr>
</thead>
</table>

**Narration:**
[t] ... narrative text on mereology type ...
[m] ... narrative text on mereology observer ...
[a] ... narrative text on mereology type constraints ...

**Formalisation:**

type
| [t] $MT = \forall (UI_i, UI_j, ..., UI_k)$
value
| [m] mereo$P : P \rightarrow MT$
axiom [Well-formedness of Domain Mereologies]
| [a] $\s
\text{a} (MT)$

The mereology type name, $MT$, chosen of course, by the domain analyser cum describer, usually properly embodies the type name, $E$, of the endurant being analysed and mereology-described. The mereology type expression $\forall (UI_i, UI_j, ..., UI_k)$ is a type expression over unique identifiers. Thus a part of type-name $P$ might be given the mereology type name MP. $\s$ (MT) is a predicate over possibly all unique identifier types of the domain description. To write down the concrete type definition for MT requires a bit of analysis and thinking
Modelling Choice 5  Mereology: As for endurant descriptions the analyser cum describer chooses for some model of a domain, one mereology, for another model of supposedly “the same” domain another mereology.

### Mereology of a Road Net

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_{Mer}$</td>
<td>$V_{UI}-set \times L_{UI}-set$</td>
</tr>
<tr>
<td>$L_{Mer}$</td>
<td>$V_{UI}-set \times H_{UI}-set$</td>
</tr>
<tr>
<td>$BC_{Mer}$</td>
<td>$B_{UI}-set$</td>
</tr>
<tr>
<td>$B_{Mer}$</td>
<td>$BC_{UI} \times R_{UI}-set$</td>
</tr>
<tr>
<td>$A_{Mer}$</td>
<td>$R_{UI}-set$</td>
</tr>
</tbody>
</table>

#### 4.3.2.1 Invariance of Mereologies

For mereologies one can usually express some invariants. Such invariants express “law-like properties”, facts which are indisputable. We refer to Sect. 4.3.4 on the next page.

### Invariance of Road Nets

Example 48  The observed mereologies must express identifiers of the state of such for road nets:

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall (vuis,luis):H_{Mer} \cdot luis \subseteq l_{uis} \wedge vuis = v_{uis}$</td>
<td>mereo_H: $H \rightarrow H_{Mer}$</td>
</tr>
<tr>
<td>$\forall (vuis,huis):L_{Mer} \cdot vuis = v_{uis} \wedge huis \subseteq h_{uis} \wedge \text{card}(huis) = 2$</td>
<td>mereo_L: $L \rightarrow L_{Mer}$</td>
</tr>
<tr>
<td>$\forall buis:H_{Mer} \cdot buis = b_{uis}$</td>
<td>mereo_B: $BC \rightarrow BC_{Mer}$</td>
</tr>
<tr>
<td>$\forall (bc_{ui},ruis):H_{Mer} \cdot bc_{ui} \in bc_{ui}s \wedge ruis = r_{uis}$</td>
<td>mereo_B: $B \rightarrow B_{Mer}$</td>
</tr>
<tr>
<td>$\forall ruis:A_{Mer} \cdot ruis = r_{uis}$</td>
<td>mereo_A: $A \rightarrow A_{Mer}$</td>
</tr>
</tbody>
</table>

160 For all hubs, $h$, and links, $l$, in the same road net, 161 if the hub $h$ connects to link $l$ then link $l$ connects to hub $h$.

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall h:H,l:L \cdot h \in hs \wedge l \in ls \Rightarrow$</td>
<td></td>
</tr>
<tr>
<td>$\text{let } (_luis) = \text{mereo}_H(h), (_huis) = \text{mereo}_L(l)$</td>
<td></td>
</tr>
<tr>
<td>$\text{in } \text{uid}_L(l) \in luis \equiv \text{uid}_H(h) \in huis$</td>
<td></td>
</tr>
</tbody>
</table>

162 For all links, $l$, and hubs, $h_a, h_b$, in the same road net, 163 if the $l$ connects to hubs $h_a$ and $h_b$, then $h_a$ and $h_b$ both connects to link $l$.

<table>
<thead>
<tr>
<th>Axiom</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall h_a,h_b:H,l:L \cdot {h_A,h_B} \subseteq hs \wedge l \in ls \Rightarrow$</td>
<td></td>
</tr>
<tr>
<td>$\text{let } (_luis) = \text{mereo}_H(h), (_huis) = \text{mereo}_L(l)$</td>
<td></td>
</tr>
<tr>
<td>$\text{in } \text{uid}_L(l) \in luis \equiv \text{uid}_H(h) \in huis$</td>
<td></td>
</tr>
</tbody>
</table>
4.3.2.2 Deductions made from Mereologies

Once we have settled basic properties of the mereologies of a domain we can, like for unique identifiers, cf. Example 42 on Page 87, “play around” with that concept: ‘the mereology of a domain’.

Possible Consequences of a Road Net Mereology

Example 49

164 are there [isolated] units from which one can not “reach” other units ?
165 does the net consist of two or more “disjoint” nets ?
166 et cetera.

We leave it to the reader to narrate and formalise the above properly.

4.3.3 Formulation of Mereologies

The observe_mereology domain descriptor, Page 90, may give the impression that the mereo type MT can be described “at the point of issue” of the observe_mereology prompt. Since the MT type expression may, in general, depend on any part sort the mereo type MT can, for some domains, “first” be described when all part sorts have had their unique identifiers defined.

4.3.4 Fixed and Varying Mereologies

The mereology of parts is not necessarily fixed.

Definition: 59 Fixed Mereology: By a fixed mereology we shall understand a mereology of a part which remains fixed over time.

Definition: 60 Varying Mereology: By a varying mereology we shall understand a mereology of a part which may vary over time.

Fixed and Varying Mereology

Example 50 Let us consider a road net, cf. Examples 38 on Page 68, 39 on Page 69, 41 on Page 72 Example 47 on the preceding page and Example 48 on the previous page. If hubs and links never change “affiliation”, that is: hubs are in fixed relation to zero one or more links, and links are in a fixed relation to exactly two hubs then the mereology of Example 47 on the preceding page is a fixed mereology. If, on the other hand hubs may be inserted into or removed from the net, and/or links may be removed from or inserted between any two existing hubs, then the mereology of Example 47 on the previous page is a varying mereology.

4.3.5 No Materials Mereology

We comment on our decision, for this monograph, to not endow materials with mereologies. A first reason is that we “restrict” the concept of mereology to part endurants, that is, to endurants with “more-or-less” fixed extents. Materials can be said to normally not have fixed extents, that is, they can “morph” from small, fixed into spatially extended forms. For domains of part-materials conjoins this is particularly true. The materials in such domains flow through and between parts. Some parts, at some times, embodying large, at other times small amounts of material. Some proper, but partial amount of material flowing from one part to a next. Et cetera. It is for the same reason that we do not endow materials with identity. So, for this monograph we decide to not suggest the modelling of materials mereologies.
4.3.6 Some Modelling Observations

It is, in principle, possible to find examples of mereologies of natural parts: rivers: their confluence, lakes and oceans; and geography: mountain ranges, flat lands, etc. But in our experimental case studies, cf. Example on Page 40, we have found no really interesting such cases. All our experimental case studies appears to focus on the mereology of artefacts. And, finally, in modelling humans, we find that their mereology encompass all other humans and all artefacts! Humans cannot be tamed to refrain from interacting with everyone and everything.

Some domain models may emphasize physical mereologies based on spatial relations, others may emphasize conceptual mereologies based on logical “connections”.

4.3.7 Conjoin Mereologies

Conjoins, their “roots” and “siblings”, enjoy some special mereology relations. Let us first consider the pragmatics of conjoins, e. Part-materials conjoins, e, are “carriers” or “holders” of materials. The carrier is e, that is, analyse_conjoin_part(e). The carried or held materials are (m₁,m₂,...,mₘ), that is, analyse_conjoin_materials(e). Usually we shall only associate more than one material with the so-called treatment conjoin. See below. The carrier or holder, p, somehow provides a “container” for each mᵢ. We shall, without loss of generality, restrict supply, pipe, valve, pump and dispose conjoins, see below, to embody just one material. Conjoins either serve to transport or to process materials. Transport is achieved by moving material between topologically connected conjoins. Processing is achieved by treating one or more materials, of the same conjoin, to interact by being operated upon. Conjoins that participate in the transport and treatment of materials, we conclude, typically form directed, acyclic nets. We shall refer to such nets as ‘conjoin nets’.

Further pragmatics are those of the interconnection of conjoins as expressed in their mereologies. First, to transport, they must form, usually directed, acyclic, nets. These nets are sequences of conjoins acting as pipes, “interspersed” by conjoins serving to fork (divert), from one, a fork conjoin, flow, into two, usually pipe, conjoins, or to join (merge) transport from two (or more) , usually pipe flow, into one, the join flow, or to treat, within a single conjoin, the ‘treatment’ conjoin, one or more materials into one or more new and/or replacement materials. By a flow net we shall understand a collection of conjoins formed as an acyclic, directed graph. Figure 4.1 abstracts a possible conjoin flow net.

![Flow Net Diagram](image-url)

*Fig. 4.1. An Abstracted Directed, Acyclic Flow Net of Conjoins*

---

9 We remind the reader of the ‘pragmatics’ paragraph of Sect. 3.13.4 on Page 56.

10 the supply, pipe, valve, pump and dispose conjoins transport are restricted to carry just one material; the treatment conjoin usually process, hence contain, more than one material.
The abstracted flow net shown in Fig. 4.1 on the preceding page is at the basis in domain models for waste management, industrial production supply, production and demand, (water oil gas, etc.) pipe lines, et cetera.

For directed, acyclic nets of material transport and treatment of units of connected conjoins we can conclude that there must be units which supply [inputs] materials to the net; units which open or close, by means of pumps [empowers] or valves [on/off], the flow of materials; units which simply pipes [flows] materials “along”; units which fork [flows] materials in two [or more]11 ongoing directions; units which join two [or more]12 material flows into one flow; units which treat [process] one or more incoming [in-flowed] and contained materials13 into one or more contained and outgoing [out-flowed] materials14; and units which dispose [outputs] one [or more]15 materials.

Let us then consider the technicalities of modelling conjoins e. The conjoin has a part: observe_conjoin_part(e), p, and it has one or more materials: observe_conjoin_materials(e), \( (m_1, m_2, \ldots, m_m) \). The mereology of p includes that of the unique identifier of e.

When we, above, cautiously, write ‘includes’ it is to say that there may be other topological or conceptual (including intentional) relations.

We can likewise consider material-parts conjoins but leave this to the reader.

This section is “conjoined” with Sect. 4.4.8 on Page 108.

### Pipeline Mereology

**Example 51** We refer to Appendix Sect. A.3.

### 4.4 Attributes

To recall: there are three sets of internal qualities: unique identifiers, part mereology and attributes. Unique identifiers and mereology are rather definite kinds of internal endurant qualities; attributes form more “free-wheeling” sets of internal qualities. Whereas, for this monograph, we suggest to not endow materials with unique identification and mereologies all endurants, i.e., including materials, are endowed with attributes.

#### 4.4.1 Inseparability of Attributes from Parts and Materials

Parts and materials are typically recognised because of their spatial form and are otherwise characterised by their intangible, but measurable attributes. That is, whereas endurants, whether discrete (as are parts) or continuous (as are materials), are physical, tangible, in the sense of being spatial [or being abstractions, i.e., concepts, of spatial endurants], attributes are intangible: cannot normally be touched, or seen, but can be objectively measured. Thus, in our quest for describing domains where humans play an active rôle, we rule out subjective “attributes”: feelings, sentiments, moods. Thus we shall abstain, in our domain science also from matters of aesthetics. We equate all endurants — which have the same type of unique identifiers, the same type of mereologies, and the same types of attributes — with one sort. Thus removing an internal

---

11 In this monograph we shall just treat the case of two fork outlets.

12 See footnote 11.

13 \( m_1, m_2, \ldots, m_m \).

14 \( m_0, m_2, \ldots, m_{m} \).

15 See footnote 11.

16 We shall not “speculate” on the possible, general relationships between \( m_1, m_2, \ldots, m_m \) and \( m_0, m_2, \ldots, m_{m} \).

17 One can see the red colour of a wall, but one touches the wall.

18 One cannot see electric current, and one may touch an electric wire, but only if it conducts high voltage can one know that it is indeed an electric wire.

19 That is, we restrict our domain analysis with respect to attributes to such quantities which are observable, say by mechanical, electrical or chemical instruments. Once objective measurements can be made of human feelings, beauty, and other, we may wish to include these “attributes” in our domain descriptions.
quality from an endurant makes no sense: the endurant of that type either becomes an endurant of another type or ceases to exist (i.e., becomes a non-entity).

We can roughly distinguish between two kinds of attributes: those which can be motivated by physical (incl. chemical) concerns, and those, which, although they embody some form of ‘physics measures’, appear to reflect on event histories: “if ‘something’, \( \phi \), has ‘happened’ to an endurant, \( e_a \), then some ‘commensurate thing’, \( \psi \), has ‘happened’ to another (one or more) endurants, \( e_b \)” where the ‘something’ and ‘commensurate thing’ usually involve some ‘interaction’ between the two (or more) endurants. It can take some reflection and analysis to properly identify endurants \( e_a \) and \( e_b \) and commensurate events \( \phi \) and \( \psi \). Example 65 on Page 112 shall illustrate the, as we shall call it, intentional pull of event histories.

4.4.2 Attribute Modelling Tools

4.4.2.1 Attribute Quality and Attribute Value

We distinguish between an attribute (as a logical proposition, of a name, i.e.) type, and an attribute value, as a value in some value space.

Analysis Function Prompt 18 **analyse_attribute_types**:

One can calculate the set of attribute type names of parts and materials with the following **domain analysis prompt**:

- \( \text{analyse_attribute_type_names} \)

Thus for a part \( p \) we may have \( \text{analyse_attribute_type_names}(p) = \{ \text{"A}_1\text{"}, \text{"A}_2\text{"}, \ldots, \text{"A}_m\text{"} \}. \)

4.4.2.2 Attribute Types and Functions

Let us recall that attributes cover qualities other than unique identifiers and mereology. Let us then consider that parts and materials have one or more attributes. These attributes are qualities which help characterise “what it means” to be a part or a material. Note that we expect every part and material to have at least one attribute. The question is now, in general, how many and, particularly, which.

Domain Description Prompt 11 **describe_attributes**:
The domain analyser experiments, thinks and reflects about endurant, \( e \), attributes. That process is initiated by the **domain description prompt**:

- \( \text{describe_attributes}(e) \)

The result of that **domain description prompt** is that the domain analyser cum describer writes down the attribute (sorts or) types and observers domain description text according to the following schema: for any endurant \( e \).

```
11. describe_attributes Observer

let \{ \text{"A}_1\text{"}, \ldots, \text{"A}_m\text{"} \} = \text{analyse_attribute_type_names}(e) in

Narration:
- [t] ... narrative text on attribute sorts ...
- [o] ... narrative text on attribute sort observers ...
- [p] ... narrative text on attribute sort proof obligations ...

Formalisation:
- type
  - [t] \( \text{A}_1, \ldots, \text{A}_m \)
  - value
    - \( \text{attr}_{\text{A}_1} : \mathcal{E} \rightarrow \text{A}_1 \), \ldots, \( \text{attr}_{\text{A}_m} : \mathcal{E} \rightarrow \text{A}_m \)
    - proof obligation [Disjointness of Attribute Types]
- [p] \( \mathcal{O} : \text{P be any part sort in [the domain description]} \)
- [p] \( \text{let } a : (\text{A}_1 | \ldots | \text{A}_m) \text{ in } \text{is}_{\text{A}_i}(a) \neq \text{is}_{\text{A}_j}(a) [i \neq i, i,j : [1..m]] \) end end

end
```

The \( \text{is}_{\text{A}_i}(e) \) is defined by \( \text{A}_i, i : [1..n] \).
Modelling Choice 6 Endurant Attributes: As for endurant and mereology descriptions the analyser
cum describer chooses for some model of a domain, one set of attributes, for another model of supposedly
“the same” domain another set of attributes ■

Let \( A_1, \ldots, A_n \) be the set of all conceivable attributes of endurants \( e: E \). (Usually \( n \) is a rather large natural
number, say in the order of a hundred conceivable such.) In any one domain model the domain analyser
cum describer selects a modest subset, \( A_1, \ldots, A_m, \) i.e., \( m < n \). Across many domain models for “more-
or-less the same” domain \( m \) varies and the attributes, \( A_1, \ldots, A_m \), selected for one model may differ from
those, \( A'_1, \ldots, A'_m' \), chosen for another model.

The type definitions: \( A_1, \ldots, A_m \), inform us that the domain analyser has decided to focus on the
distinctly named \( A_1, \ldots, A_m \) attributes. \(^{20}\) The value clauses \( \text{attr}_{A_1} : P \rightarrow A_1, \ldots, \text{attr}_{A_m} : P \rightarrow A_m \) are then “auto-
matically” given: if an endurant, \( e : E \), has an attribute \( A_i \), then there is postulated, “by definition” [eureka]
an attribute observer function \( \text{attr}_{A_i} : E \rightarrow A_i \) et cetera ■

We cannot automatically, that is, syntactically, guarantee that our domain descriptions secure that the
various attribute types for a endurant sort denote disjoint sets of values. Therefore we must prove it.

4.4.2.3 Attribute Categories

Michael A. Jackson [252] has suggested a hierarchy of attribute categories: from static to dynamic values –
and within the dynamic value category: inert values, reactive values, active values – and within the dynamic
active value category: autonomous values, biddable values and programmable values. We now review these
attribute value types. The review is based on [252, M.A.Jackson]. Endurant attributes are either constant or
varying, i.e., static or dynamic attributes.

Attribute Category: 1 By a \textbf{static attribute}, \( a : A \), we shall understand an
attribute whose values are constants, i.e., cannot change.

\textbf{Example 52} Let us exemplify road net attributes in this and the next examples. And let us assume the
following attributes: year of first link construction and link length at that time. We may consider both to
be static attributes: The year first established, seems an obvious static attribute and the length is fixed at
the time the road was first built.

Attribute Category: 2 By a \textbf{dynamic attribute}, \( a : A \), we shall under-
stand an attribute whose values are variable, i.e., can change. Dynamic attributes are either \textit{inert, reactive}
or \textit{active} attributes.

\textbf{Example 53} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

Attribute Category: 3 By an \textbf{inert attribute}, \( a : A \), we shall understand a
dynamic attribute whose values only change as the result of external stimuli where these stimuli prescribe
new values.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 53} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.

\textbf{Example 54} And let us now further assume the following link attribute: link name. We may consider
it to be an inert attribute: the name is not “assigned” to the link by the link itself, but probably by some
road net authority which we are not modelling.
4.4 Attributes

**Reactive Attributes**

**Example 54** Let us further assume the following two link attributes: “wear and tear”, respectively “icy and slippery”. We will consider those attributes to be reactive in that automobiles (another part) travelling the link, an external “force”, typically causes the “wear and tear”, respectively the weather (outside our domain) causes the “icy and slippery” property.

**Attribute Category: 5** By an active attribute, \( a: A \), \( \text{is active attribute}(a) \), we shall understand a dynamic attribute whose values change (also) of its own volition. Active attributes are either autonomous, biddable or programmable attributes.

**Attribute Category: 6** By an autonomous attribute, \( a: A \), \( \text{is autonomous attribute}(a) \), we shall understand a dynamic active attribute whose values change only “on their own volition”. The values of an autonomous attributes are a “law onto themselves and their surroundings”.

**Autonomous Attributes**

**Example 55** We enlarge scope of our examples of attribute categories to now also include automobiles (on the road net). In this example we assume that an automobile is driven by a human [behaviour]. These are some automobile attributes: velocity, acceleration, and moving straight, or turning left, or turning right. We shall consider these three attributes to be autonomous. It is the driver, not the automobile, who decides whether the automobile should drive at constant velocity, including 0, or accelerate or decelerate, including stopping. And it is the driver who decides when to turn left or right, or not turn at all.

**Attribute Category: 7** By a biddable attribute, \( a: A \), \( \text{is biddable attribute}(a) \) we shall understand a dynamic active attribute whose values are prescribed but may fail to be observed as such.

**Attribute Category: 8** By a programmable attribute, \( a: A \), \( \text{is programmable attribute}(a) \), we shall understand a dynamic active attribute whose values can be prescribed.

**Programmable Attribute**

**Example 56** We continue with the automobile on the road net examples. In this example we assume that an automobile includes, as one inseparable entity, “the driver”. These are some automobile attributes: position on a link, velocity, acceleration (incl. deceleration), and direction: straight, turning left, turning right. We shall now consider these three attributes to be programmable.

Figure 4.2 captures an attribute value ontology.

![Attribute Value Ontology](image_url)
Figure 4.2 on the preceding page hints at three categories of dynamic attributes: **monitorable only**, **biddable** and **programmable** attributes.

**Attribute Category:** 9 By a **monitorable only attribute**, a:A, **is_monitorable_only_attribute(a)**, we shall understand a dynamic active attribute which is either inert or reactive or autonomous.

That is: 
$$
\text{is_monitorable(e)} \equiv \text{is_inert(e)} \lor \text{is_reactive(e)} \lor \text{is_autonomous(e)}.
$$

### Road Net Attributes

**Example 57** We treat some attributes of the hubs of a road net.

167 There is a hub state. It is a set of pairs, \((l_f, l_t)\), of link identifiers, where these link identifiers are in the mereology of the hub. The meaning of the hub state in which, e.g., \((l_f, l_t)\) is an element, is that the hub is open, “green”, for traffic from link \(l_f\) to link \(l_t\). If a hub state is empty then the hub is closed, i.e., “red” for traffic from any connected links to any other connected links.

168 There is a hub state space. It is a set of hub states. The current hub state must be in its state space.

The meaning of the hub state space is that its states are all those the hub can attain.

169 Since we can think rationally about it, it can be described, hence we can model, as an attribute of hubs, a history of its traffic: the recording, per unique bus and automobile identifier, of the time ordered presence in the hub of these vehicles. Hub history is an event history.

### Invariance of Road Net Traffic States

**Example 58** We continue Example 57.

170 The link identifiers of hub states must be in the set, \(l_{\text{ui}s}\), of the road net’s link identifiers.

**axiom**

\[
\forall h \in h \cdot \text{obs}_{H \Sigma}(h) \in \text{obs}_{H \Omega}(h)
\]

**type**

\[
H \Sigma = (L_{\text{UI}} \times L_{\text{UI}})\text{-set}
\]

**time ordered:**

\[
\text{time ordered}(ht) \equiv \ldots
\]

### Pipeline Attributes

**Example 59** We refer to Appendix Sect. A.4.

You may skip Example 60 in a first reading.

### Road Transport: Further Attributes

**Example 60** **Links:** We show just a few attributes.

171 There is a link state. It is a set of pairs, \((h_f, h_t)\), of distinct hub identifiers, where these hub identifiers are in the mereology of the link. The meaning of a link state in which \((h_f, h_t)\) is an element is that the link is open, “green”, for traffic from hub \(h_f\) to hub \(h_t\). Link states can have either 0, 1 or 2 elements.
There is a link state space. It is a set of link states. The meaning of the link state space is that its states are all those the which the link can attain. The current link state must be in its state space. If a link state space is empty then the link is (permanently) closed. If it has one element then it is a one-way link. If a one-way link, \( l \), is imminent on a hub whose mereology designates that link, then the link is a “trap”, i.e., a “blind cul-de-sac”.

Since we can think rationally about it, it can be described, hence it can model, as an attribute of links a history of its traffic: the recording, per unique bus and automobile identifier, of the time ordered positions along the link (from one hub to the next) of these vehicles.

The hub identifiers of link states must be in the set, \( h_{ui} \), of the road net’s hub identifiers.

There are two notions of time at play here: the indefinite “real” or “actual” time; and the definite calendar, hour, minute and second time designation occurring in some textual form in, e.g., time tables.

Bus companies: operate a number of lines that service passenger transport along routes of the road net. Each line being serviced by a number of buses.

Bus companies create, maintain, revise and distribute [to the public (not modeled here), and to buses] bus time tables, not further defined.

Buses: We show just a few attributes.

There are two notions of time at play here: the indefinite “real” or “actual” time; and the definite calendar, hour, minute and second time designation occurring in some textual form in, e.g., time tables.

Buses occupy positions on the road net:

- either at a hub identified by some \( h_{ui} \),
- or on a link, some fraction, \( f:Fract \) down an identified link, \( l_{ui} \), from one of its identified connecting hubs, \( fh_{ui} \), in the direction of the other identified hub, \( th_{ui} \).
178 BPos = atHub | onLink [programmable, Df.8 Pg.97]
178a atHub :: hui:HUI
178b onLink :: fhui:HUI × lui:LUI × frac:Fract × thui:HUI
178b Fract = Real, axiom frac:Fract • 0 < frac < 1
179 ...
value
177 attr_BusTimTbl: B → BusTimTbl
178 attr_BPos: B → BPos

Private Automobiles: We show just a few attributes: .................................................................

We illustrate but a few attributes:

180 Automobiles have static number plate registration numbers.
181 Automobiles have dynamic positions on the road net:
  [178a] either at a hub identified by some hui,
  [178b] or on a link, some fraction, frac:Fract down an identified link, lui, from one of its identified con-
necting hubs, fhui, in the direction of the other identified hub, thui.

type
180 RegNo [static, Df.1 Pg.96]
181 APos = atHub | onLink [programmable, Df.8 Pg.97]
178a atHub :: hui:HUI
178b onLink :: fhui:HUI × lui:LUI × frac:Fract × thui:HUI
178b Fract = Real, axiom frac:Fract • 0 < frac < 1
value
180 attr_RegNo: A → RegNo
181 attr_APos: A → APos

Obvious attributes that are not illustrated are those of velocity and acceleration, forward or backward movement, turning right, left or going straight, etc. The acceleration, deceleration, even velocity, or turning right, turning left, moving straight, or forward or backward are seen as command actions. As such they denote actions by the automobile — such as pressing the accelerator, or lifting accelerator pressure or braking, or turning the wheel in one direction or another, etc. As actions they have a kind of counterpart in the velocity, the acceleration, etc. attributes.

Observe that bus companies each have their own distinct bus time table, and that these are modeled as programmable, Item 175 on the previous page, Page 99. Observe then that buses each have their own distinct bus time table, and that these are model-led as inert, Item 177 on the preceding page, Page 99. In Items 270–271b Pg. 149 we shall see how the buses communicate with their respective bus companies in order for the buses to obtain the programmed bus time tables “in lieu” of their inert one! In Items 169 Pg. 98 and 173 Pg. 99, we illustrated an aspect of domain analysis & description that may seem, and at least some decades ago would have seemed, strange: namely that if we can think, hence speak, about it, then we can model it “as a fact” in the domain. The case in point is that we include among hub and link attributes their histories of the timed whereabouts of buses and automobiles.21

Calculating Attributes

182 Given endurant e we can meta-linguistically calculate names for its static attributes.
183 Given endurant e we can meta-linguistically calculate name for its monitorable-only attributes attributes.
184 Given endurant e we can meta-linguistically calculate names for its controllable attributes.
185 These four sets make up all the attributes of endurant e.

The type names ST, MA, PT designate mutually disjoint sets, ST, of names of static attributes, sets, MA, of names of monitorable, i.e., monitorable-only and biddable, attributes, sets, PT, of names of programmable, i.e., fully controllable attributes.

21 By using the term meta-linguistically here we shall indicate that we go outside what is computable – and thus appeal to the reader’s forbearance.
4.4 Attributes

value
182  stat_attr_types: E \rightarrow ST
183  moni_attr_types: E \rightarrow MA
184  prgr_attr_types: E \rightarrow PT

axiom
185  \forall e:E \cdot
186  let stat_nms = stat_attr_types(e),
187  moni_nms = moni_attr_types(e),
188  prgr_nms = prgr_attr_types(e) in
189  card stat_nms + card moni_nms + card prgr_nms
190  = card(stat_nms \cup moni_nms \cup prgr_nms) end

The above formulas are indicative, like mathematical formulas, they are not computable.

186  Given endurant e we can meta-linguistically calculate its static attribute values, stat_attr_vals;
187  given endurant e we can meta-linguistically calculate its monitorable-only attribute values, moni_attr_vals;
188  given endurant e we can meta-linguistically calculate its programmable attribute values, prgr_attr_vals.

The type names sa1, ..., pap refer to the types denoted by the corresponding types name nsa1, ..., npap.

value
186  stat_attr_vals: E \rightarrow SA1 \times SA2 \times \ldots \times SAS
186  stat_attr_vals(e) \equiv
186  let \{nsa1,nsa2,\ldots,nsas\} = stat_attr_types(e) in
186  (attr_{sa1}(e),attr_{sa2}(e),\ldots,attr_{sas}(e)) end

187  moni_attr_vals: E \rightarrow MA1 \times MA2 \times \ldots \times MAm
187  moni_attr_vals(e) \equiv
187  let \{nma1,nma2,\ldots,nmam\} = moni_attr_types(e) in
187  (attr_{ma1}(e),attr_{ma2}(e),\ldots,attr_{mam}(e)) end

188  prgr_attr_vals: E \rightarrow PA1 \times PA2 \times \ldots \times PAp
188  prgr_attr_vals(e) \equiv
188  let \{npa1,npa2,\ldots,npap\} = prgr_attr_types(e) in
188  (attr_{pa1}(e),attr_{pa2}(e),\ldots,attr_{pap}(e)) end

The “ordering” of type values, \(\text{attr}_{sa1}(e),\ldots,\text{attr}_{sas}(e)\), \(\text{attr}_{ma1}(e),\ldots,\text{attr}_{mam}(e)\), et cetera, is arbitrary.

4.4.3 A Discourse on Attribute Kinds

In this, in a sense, discursive, section we shall depart, somewhat, from a more direct presentation of analysis and description prompts. We shall muse, as it were, about the following, perceived kinds of concepts and attributes:

- space, time and substance,
- spatio and temporal attributes,
- natural and artefactual attributes,
- geometric attributes,
- action and event attributes,
- and others!
**Space, Time and Matter**[^23]: Space, SPACe, time, TIME, and substance, MATTER, cannot be solely attributes of endurants[^24]. Endurants exist in space and time. Manifest endurants have substance, i.e., consists of MATTER. These three kinds of properties follow by transcendental deduction from rational reasoning. So it is futile to ascribe attributes solely of these kinds to endurants! But, stop here, pause a bit. Somehow we must ascribe what appears to be space, time and substance properties to endurants: length, speed, weight. So what is the problem? The problem is that these latter kinds of properties are artefactual properties. Mankind have found a need to somehow measure spatial, temporal and substance phenomena.

**Spatio-like Attributes:** The geographical location of a specific “point”[^25] on the surface of earth, represented by its longitude[^26] and latitude[^27], can be an attribute of an endurant. This is so because the representation are artefactual qualities, not transcendentally deducible facts. POINT are mathematical concepts, created as mathematical abstractions – as are LINEs, CURVES, SURFACEs and EXTENTS. The respective Lengths, Areas and Volumes of spatial entities are artefactual qualities ascribed by humans and measured in, for example, m, m² and m³, respectively.

**Temporal-like Attributes:** Other attributes are endowed, as properties of endurants, not by man, but inherently there, given to us. The Time at which some action is invoked or some event occurs, is not a TIME; and the TimeInterval (time interval or duration) between two actions or events is not a TI. Instead humans, after many attempts, have devised ways and means of representing, respectively measuring Times and TimeInterval[^28]. A DAY is not 24 hours, 0 minute, 0 seconds, and which or whatever fraction of a second you may think of. A SIDERIAL YEAR is the TI it takes for the earth to orbit the sun. While doing so, the earth spins on its axis. One complete spin takes exactly a DAY. But our concept of a Day is that of 24 Hours, with each hour “divided” into 60 Minutes, and each minute into 60 Seconds, and so forth. So Year, Month, Week, Day, Hour, Minute, Second, etc., are human constructions devised to represent time intervals. These time interval representations are of some absolute kind. They are independent of which endurant, at which POINT in SPACe, e.g., where on earth, they may be related to. As human constructions they have lead to many ingenious means for their measure: clocks of many kind[^29],[^30]. But clocks do not measure time, only time intervals. The time that an event occurred, or to occur, or an action was (to be) invoked, are of absolute kind. A time, like “Saturday 16 May, 2020, at 9:27:03” is such an example. Time has, to some physicists, no [absolute] beginning point. There is no “On the first day and in the first hour of creation”. So mankind has settled on some, you may say, ‘compromise’. We “speak of” time, usually, as if it was an interval: “Around the year 482 Before Christ[^31]”, or “July 6, 2020: 10:03 am, after Christ[^32]”. Time indications must state the, or an approximate location on earth, for which they are given, or some other reference frame, e.g., a time zone, such as Greenwich Mean Time, GMT or

[^23]: We remind the reader of Chapter 2.
[^24]: But spatial measures, time stamps, time intervals, and substance (matters) may be attributes
[^25]: By “a specific ‘point’ ” we do not mean a POINT.
[^26]: Longitude is the angle east or west of a reference meridian to another meridian that passes through that point [Wikipedia].
[^27]: Latitude is the angle between the equatorial plane and the straight line that passes through that point and through (or close to) the center of the earth [Wikipedia]
[^28]: To wit: quartz-crystal clocks of the 1930s.
[^29]: A second is defined as 9,192,631,770 oscillations of the caesium atom, off by only one second after running for 300 million years.
[^30]: Some references [www.encyclopedia.com]:

- Gibbs, Sharon L. Greek and Roman Sundials.

[^31]: – time of the birth of the Greek philosopher Plato
[^32]: – time at last editing this text
other (CET, EET, WET, ET, PST etc.). For early continental explorers and ocean sea-farers, accurate chronometers became indispensable\textsuperscript{33}.

**Spatio-Temporal-like Attributes:** We talk, for example, of speed as distance, i.e., length, covered by time interval. And we talk, for example, of velocity, i.e., speed with vectorial direction. On one hand they are spatio-temporal phenomena, inherent as transcendently deducible facts. On the other hand we also experience them, and may have need to represent them as attributes. In other words, we must be careful in our analysis.

**Action and Event Attributes**\textsuperscript{34}: An important class of attributes record actions and events that occur to endurants. That an action or event occurs or has occurred is immaterial, but we can talk about it! As such it may need being recorded in an appropriate attribute. Such recording are, to be meaningful, time-stamped.

### Action and Event Attributes

**Example 61** From our continuing road transport example we give an example. The occurrence of an automobile at a hub or on a link is an event and can, as such, be recorded in both hub, link and automobile “history” attributes. From our likewise continuing pipeline example we give examples. The actions of opening and closing of valves, the actions of starting and ending of pumping, and the events of a pipe unit becoming empty, or overflowing (“choke”), or changing from laminar to turbulent flow\textsuperscript{35}, can, as such, be recorded in both pipeline unit “history” attributes.

**Natural and Artefactual Attributes:** From the above we can see that in seeking properties of endurants we waver between natural phenomena and artefactual “measures”. So we need be clear of the distinction, for an endurant, whether what may seem to be [a kind of] an endurant attribute is really a SPACE, TIME or SUBSTANCE phenomenon; or whether it is one for which we have “invented” measures. The former we refer to as natural attributes. They can be expressed using the physical attribute kinds detailed in Sect. 4.4.4 next The latter we refer to as artefactual attributes. They can be expressed using both physical attribute kinds and domain concepts such as, for example, unique identifiers.

**Geometrical Attributes:** Manifest endurants reside in space. But we characterise them by such geometric measures as position in some earthly, or relative coordinate system, length, a relative measure, and volume. For intricate geometric objects we may be comprehension-wise better off by presenting them in diagrams, drawings or annotated photos or videos.

### 4.4.3.2 A Preliminary Conclusion

We could go on finding further varieties of attributes. But we stop here! So why this section? So that you may hopefully be very careful in your assignment of attributes.

### 4.4.4 Physics Attributes

In this section we shall muse about the kind of attributes that are typical of natural parts, but which may also be relevant as attributes of artefacts.

Typically, when physicists write computer programs, intended for calculating physics behaviours, they “lump” all of these into the type Real, thereby hiding some important physics ‘dimensions’. In this section we shall review that which is missing!

The subject of physical dimensions in programming languages is rather decisively treated in David Kennedy’s 1996 PhD Thesis [259] — so there really is no point in trying to cast new light on this subject other than to remind the reader of what these physical dimensions are all about.

\textsuperscript{33} Christiaan Huygens, following his invention of the pendulum clock in 1656, made the first attempt at a marine chronometer in 1673

\textsuperscript{34} We refer to Defns. 65 on Page 127 and 66 on Page 127 for definitions of the concepts of action and event.

\textsuperscript{35} Becoming empty, overflowing or transiting between laminar and turbulent flows are fuzzy measures; see Sect. 4.4.9 on Page 109.
4.4.4.1 SI: The International System of Quantities

In physics we operate on values of attributes of manifest, i.e., physical phenomena. The type of some of these attributes are recorded in well known tables, cf. Tables 4.1–4.3. Table 4.1 shows the base units of physics.

<table>
<thead>
<tr>
<th>Base quantity</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>meter</td>
<td>m</td>
</tr>
<tr>
<td>mass</td>
<td>kilogram</td>
<td>kg</td>
</tr>
<tr>
<td>time</td>
<td>second</td>
<td>s</td>
</tr>
<tr>
<td>electric current</td>
<td>ampere</td>
<td>A</td>
</tr>
<tr>
<td>thermodynamic temp</td>
<td>kelvin</td>
<td>K</td>
</tr>
<tr>
<td>amount of substance</td>
<td>mole</td>
<td>mol</td>
</tr>
<tr>
<td>luminous intensity</td>
<td>candela</td>
<td>cd</td>
</tr>
</tbody>
</table>

Table 4.1. Base SI Units

Table 4.2 shows the units of physics derived from the base units. Table 4.3 shows further units of physics.

<table>
<thead>
<tr>
<th>Name</th>
<th>Derived Quantity</th>
<th>Derived Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>radian</td>
<td>rad angle</td>
<td>m/m</td>
</tr>
<tr>
<td>steradian</td>
<td>sr solid angle</td>
<td>m²×m⁻²</td>
</tr>
<tr>
<td>Hertz</td>
<td>Hz frequency</td>
<td>s⁻¹</td>
</tr>
<tr>
<td>newton</td>
<td>N force, weight</td>
<td>kg×m×s⁻²</td>
</tr>
<tr>
<td>pascal</td>
<td>Pa pressure, stress</td>
<td>N/m²</td>
</tr>
<tr>
<td>joule</td>
<td>J energy, work, heat</td>
<td>N×m</td>
</tr>
<tr>
<td>watt</td>
<td>W power, radiant flux</td>
<td>J/s</td>
</tr>
<tr>
<td>coulomb</td>
<td>C electric charge</td>
<td>s×A</td>
</tr>
<tr>
<td>volt</td>
<td>V electromotive force</td>
<td>W/A (kg×m²×s⁻³×A⁻¹)</td>
</tr>
<tr>
<td>farad</td>
<td>F capacitance</td>
<td>C/V (kg⁻¹×m⁻²×s⁻²×A²)</td>
</tr>
<tr>
<td>ohm</td>
<td>Ω electrical resistance</td>
<td>V/A (kg×m²×s²×A²)</td>
</tr>
<tr>
<td>siemens</td>
<td>S electrical conductance</td>
<td>A/V (kg×m²×s³×A²)</td>
</tr>
<tr>
<td>weber</td>
<td>Wb magnetic flux</td>
<td>V×s (kg×m²×s⁻²×A⁻¹)</td>
</tr>
<tr>
<td>tesla</td>
<td>T magnetic flux density</td>
<td>Wb/m² (kg×s²×A⁻¹)</td>
</tr>
<tr>
<td>henry</td>
<td>H inductance</td>
<td>Wb/A (kg×m²×s²×A²)</td>
</tr>
<tr>
<td>degree Celsius</td>
<td>°C temp. ref. to 273.15 K</td>
<td>K</td>
</tr>
<tr>
<td>lumen</td>
<td>lm luminous flux</td>
<td>cd×sr (cd)</td>
</tr>
<tr>
<td>lux</td>
<td>lx illuminance</td>
<td>lm/m² (m²×cd)</td>
</tr>
</tbody>
</table>

Table 4.2. Derived SI Units

derived from the base units. velocity is speed with three dimensional direction and is, for example, given as

- velocity, meter per second with direction: \( m/s \)
- acceleration, meter per second squared and \( (longitude,latitude,azimuth) \) measured in radian: \( m/s²(r,r,r) \)

Table 4.4 shows standard prefixes for SI units of measure and Tables 4.5 show fractions of SI units.

The point in bringing this material is that when modelling, i.e., describing domains we must be extremely careful in not falling into the trap of modelling physics types, etc., as we do in programming – by simple Real\(\). We claim, without evidence, that many trivial programming mistakes are due to confusions between especially derived SI units, fractions and prefixes.
### Table 4.3. Further SI Units

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
<th>Derived Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>square meter</td>
<td>m²</td>
</tr>
<tr>
<td>volume</td>
<td>cubic meter</td>
<td>m³</td>
</tr>
<tr>
<td>speed</td>
<td>meter per second</td>
<td>m/s</td>
</tr>
<tr>
<td>wave number</td>
<td>reciprocal meter</td>
<td>m⁻¹</td>
</tr>
<tr>
<td>mass density</td>
<td>kilogram per cubic meter</td>
<td>kg/m³</td>
</tr>
<tr>
<td>specific volume</td>
<td>cubic meter per kilogram</td>
<td>m³/kg</td>
</tr>
<tr>
<td>current density</td>
<td>ampere per square meter</td>
<td>A/m²</td>
</tr>
<tr>
<td>magnetic field strength</td>
<td>ampere per meter</td>
<td>A/m</td>
</tr>
<tr>
<td>substance concentration</td>
<td>mole per cubic meter</td>
<td>mol/m³</td>
</tr>
<tr>
<td>luminance</td>
<td>candela per square meter</td>
<td>cd/m²</td>
</tr>
<tr>
<td>mass fraction</td>
<td>kilogram per kilogram</td>
<td>kg/kg = 1</td>
</tr>
</tbody>
</table>

### Table 4.4. Standard Prefixes for SI Units of Measure

<table>
<thead>
<tr>
<th>Prefix name</th>
<th>Prefix symbol</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>deca</td>
<td>da</td>
<td>10⁰</td>
</tr>
<tr>
<td>hecto</td>
<td>h</td>
<td>10¹</td>
</tr>
<tr>
<td>kilo</td>
<td>k</td>
<td>10²</td>
</tr>
<tr>
<td>mega</td>
<td>M</td>
<td>10⁶</td>
</tr>
<tr>
<td>giga</td>
<td>G</td>
<td>10⁹</td>
</tr>
<tr>
<td>tera</td>
<td>T</td>
<td>10¹²</td>
</tr>
<tr>
<td>peta</td>
<td>P</td>
<td>10¹⁵</td>
</tr>
<tr>
<td>exa</td>
<td>E</td>
<td>10¹⁸</td>
</tr>
<tr>
<td>zetta</td>
<td>Z</td>
<td>10²¹</td>
</tr>
<tr>
<td>yotta</td>
<td>Y</td>
<td>10²⁴</td>
</tr>
<tr>
<td>deci</td>
<td>d</td>
<td>10⁻¹</td>
</tr>
<tr>
<td>centi</td>
<td>c</td>
<td>10⁻²</td>
</tr>
<tr>
<td>milli</td>
<td>m</td>
<td>10⁻³</td>
</tr>
<tr>
<td>micro</td>
<td>µ</td>
<td>10⁻⁶</td>
</tr>
<tr>
<td>nano</td>
<td>n</td>
<td>10⁻⁹</td>
</tr>
<tr>
<td>pico</td>
<td>p</td>
<td>10⁻¹²</td>
</tr>
<tr>
<td>femto</td>
<td>f</td>
<td>10⁻¹⁵</td>
</tr>
<tr>
<td>atto</td>
<td>a</td>
<td>10⁻¹⁸</td>
</tr>
<tr>
<td>zepto</td>
<td>z</td>
<td>10⁻²¹</td>
</tr>
<tr>
<td>yocto</td>
<td>y</td>
<td>10⁻²⁴</td>
</tr>
</tbody>
</table>

### Table 4.5. SI Units of Measure and Fractions

#### 4.4.4.2 Units are Indivisible

A volt, kg $\times$ m$^2$ $\times$ s$^{-3}$ $\times$ A$^{-1}$, see Table 4.2, is “indivisible”. It is not a composite structure of mass, length, time, and electric current – in some intricate relationship.

•••

Physical attributes may ascribe mass and volume to endurants. But they do not reveal the substance, i.e., the material from which the endurant is made. That is done by chemical attributes.

#### 4.4.4.3 Chemical Elements

The chemical elements are, to us, what makes up a substance of MATTER. The mole, mol, substance is about chemical molecules. A mole contains exactly $6.02214076 \times 10^{23}$ (the Avogadro number) constituent particles, usually atoms, molecules, or ions – of the elements, cf. 'The Periodic Table', en.wikipedia.org/wiki/Periodic_table, cf. Fig. 4.3. Any specific molecule is then a compound of two or more elements, for example, calciumphosphat: Ca₃(PO₄)₂.
Moles bring substance to endurants. The physics attributes may ascribe weight and volume to endurants, but they do not explain what it is that gives weight, i.e., fills out the volume.

### Example 62

**Road Net**

**Hub attributes:**
- number of lanes,
- surface,
- etc.;

**Link attributes:**
- number of lanes,
- surface.

**Automobile attributes:**
- Length
- Width
- Height

### 4.4.5 Presentation of Physical Attributes

Physical attributes have several dimensions [i] First, as an example, there are the abstract physical units time interval, distance, mass, etc. [ii] Then, to continue with these units, there are the concrete physical units,
e.g., s (second), e.g., m (meter) and, e.g., g (gramme). [iii] Finally there are the scales $10^n$, $n$ a positive natural number, or $n$ a negative such. We suggest that your abstract physical attribute type, $A$, embodies

202 of what abstract physical units it is, i.e., $\text{obs}_\text{phys}\text{unit}$.

203 of what concrete physical units it is, i.e., $\text{obs}_\text{conr}\text{unit}$, and

204 its scale, i.e., $\text{obs}_\text{scale}$.

\begin{verbatim}

\text{type}
\text{AbsPhysUnitAttr} = "\text{time_interval}" | "\text{length}" | "\text{mass}" | ...
\text{ConcPhysUnitAttr} = "\text{second}" | "\text{minute}" | "\text{hour}" | "\text{meter}" | "\text{gram}" | ...
\text{PhysScaleAttr} = \text{Intg}
\end{verbatim}

We suggest that your abstract physical attribute type, $A$, embodies

202 of what abstract physical units it is, i.e., $\text{obs}_\text{phys}\text{unit}$.

203 of what concrete physical units it is, i.e., $\text{obs}_\text{conr}\text{unit}$, and

204 its scale, i.e., $\text{obs}_\text{scale}$.

These sketched observer functions are partial as they are undefined for non-physical attributes.

\subsection*{4.4.6 The Care and Feeding of Physical Attributes}

The above, i.e., Sect. 4.4.5, suggests that we introduce the following analysis predicates and functions:

\begin{verbatim}

\text{is\_physical\_attribute}: A \rightarrow \text{Bool},
\text{analyse\_abs\_phys\_attr}: A \rightarrow \text{Abs\_Phys\_Attr},
\text{analyse\_conc\_phys\_attr}: A \rightarrow \text{Conc\_Phys\_Attr},
\text{analyse\_phys\_attr\_scale}: A \rightarrow \text{Phys\_Attr\_Scale},
\end{verbatim}

where the user then defines the concrete $\text{Abs\_Phys\_Attr}$, $\text{Conc\_Phys\_Attr}$ and $\text{Phys\_Attr\_Scale}$ types as per Sect. 4.4.5. Then we suggest that the user define a number of “conversion” functions:

- \text{convert\_from\_to(from\_concrete\_unit, to\_concrete\_unit)}: converts between concrete physical attributes, e.g., pounds and kilograms, meters and yards, meters and kilometers, gram and ounces, etc.,
- \text{add\_concrete\_values(v1, v2)}, \text{subtract\_concrete\_values(v1, v2)}, \text{multiply\_concrete\_values(v1, v2)}, etc. — you see what we mean!

We suggest that the domain analyser & describer, when professionally developing domain models for domains that can be characterised by some dominance of physical attribute endurants, be very careful in caring for their physical unit analysis & description. Many aircraft, train and power plant disasters can be referred back to software which handles physical units erroneously. We refer to [6, 74, 181] for more on this issue.

\subsection*{4.4.7 Artefactual Attributes}

\subsubsection*{4.4.7.1 Examples of Artefactual Attributes}

We exemplify some artefactual attributes.

- **Designs.** Artefacts are man-made endurants. Hence “exhibit” a design. My three dimensional villa has floor plans, etc. The artefact attribute: ‘design’ can thus be presented by the architect’s or the construction engineer’s CAD/CAM drawings.

- **States** of an artefact, such as, for example, a road intersection (or railway track) traffic signal; and

- **Currency**, e.g., Kr, $, €, ¥, etc., used as an attribute\footnote{One could also consider a [10 €] bank note to be an artefact, i.e., a part.}, say the cost of a train ticket.

- **Artefactual Dimensions.** Let the domain be that of industrial production whose attributes could then be: production: units produced per year, $\text{Units/Year}$; growth: increase in units produced per year, $\text{Units \times Year}^{-2}$; productivity: production per staff, $\text{Units \times Year}^{-1} \times \text{Staff}^{-1}$ — where the base for units and staff are natural numbers.
Document Artefactual Attributes

Example 63  Let us consider documents as artefactual parts. Typical document attributes are: (i) kind of document: book, report, pamphlet, letter and ticket; (ii) publication date; (iii) number of pages; (iv) author/publisher and (v) possible colophon information. All of these attributes are non-physics quantities.

Road Net Artefactual Attributes

Example 64  Hub attributes:
209 state: set of pairs of link identifiers from, respectively to which automobiles may traverse the hub;
210 state space: set of all possible hub states.

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>209. HΣ = (L1×L1)-set</td>
<td>209. attr,HΣ:H→HΣ</td>
</tr>
<tr>
<td>210. HΩ = HΣ-set</td>
<td>210. attr,HΩ:H→HΩ</td>
</tr>
</tbody>
</table>

Link attributes:
211 state: set of 0, 1, 2 or 3 pairs of adjacent hub identifiers, the link is closed, open in one direction (closed in the opposite), open in the other direction, or open in both directions; and
212 state space: set of all possible link states.

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>211. LΣ = (L1×L1)-set</td>
<td>211. attr,LΣ:L→LΣ</td>
</tr>
<tr>
<td>212. LΩ = LΣ-set</td>
<td>212. attr,LΩ:L→LΩ</td>
</tr>
</tbody>
</table>

4.4.8 Conjoin Attributes

This section is “conjoined” with Sect. 4.3.7 on Page 93. Part-materials conjoins, the atomic part and its one or more materials, enjoy some special attributes relations. We refer to Sect. 4.3.7’s Fig. 4.1 on Page 93. We observe there the following generic flow-net conjoins: supply, pipe, pump, valve, join, fork, treat and dispose. For these we now suggest some archetypical conjoin part attributes. Indices i index join inlets: 1, 2 or more, and fork outlet: 1, 2, or more. If index is left out the conjoin unit has at most 1 inlet and at most one outlet.

4.4.8.1 Conjoin Attribute Categories

<table>
<thead>
<tr>
<th>attr</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Substance</td>
<td>the substance name of the material that can be “carried” by the conjoin part unit.</td>
</tr>
<tr>
<td>Volume</td>
<td>volume of material that the conjoin part can take, measured say in m³;</td>
</tr>
<tr>
<td>Max_In_Flow</td>
<td>typically a volume/sec quantity, measured as max_in_flow_i : m³/sec;37</td>
</tr>
<tr>
<td>Max_Out_Flow</td>
<td>as for in-flow, but now for outlets: max_out_flow_i : m³/sec;38</td>
</tr>
<tr>
<td>Curr_In_Flow</td>
<td>as for volume/sec quantity, measured as curr_in_flow_i : m³/sec;</td>
</tr>
<tr>
<td>Curr_Out_Flow</td>
<td>as for in-flow, but now for outlets: curr_out_flow_i : m³/sec;</td>
</tr>
<tr>
<td>Num_Inlets</td>
<td>a natural number n, typically 1 or 2;</td>
</tr>
<tr>
<td>Num_Outlets</td>
<td>a natural number n, typically 1 or 2;</td>
</tr>
<tr>
<td>Open_Close</td>
<td>of a valve or pump, e.g., indicated as &quot;open&quot; or &quot;closed&quot;;</td>
</tr>
</tbody>
</table>

37 set by conjoin unit manufacturer to indicate maximum laminar flow
38 set by conjoin unit manufacturer to indicate maximum laminar flow
4.4.8.2 Conjoin Attribute Assignments

**supply:** We assume a well of indefinite capacity.
- attr\_Substance\[s\], attr\_Max\_Out\_Flow\_M\_i, attr\_Curr\_Out\_Flow\_M\_i.
- Possibly other attributes.

**pipe:** We assume a pipe to be as a tube.
- attr\_Substance, attr\_Volume\_Substance, attr\_Max\_In\_Flow, attr\_Curr\_In\_Flow, attr\_Max\_Out\_Flow, attr\_Curr\_Out\_Flow.
- Possibly other attributes.

**pump:** We assume a simple positive displacement pump.
- attr\_Substance, attr\_Volume\_Substance, attr\_Max\_In\_Flow, attr\_Curr\_In\_Flow, attr\_Max\_Out\_Flow, attr\_Curr\_Out\_Flow.
- Possibly other attributes.

**valve:** We assume a simple butterfly valve.
- attr\_Substance, attr\_Volume\_Substance, attr\_Max\_In\_Flow, attr\_Curr\_In\_Flow, attr\_Max\_Out\_Flow, attr\_Curr\_Out\_Flow, attr\_Open\_Close.
- Possibly other attributes.

**join:** A join has 1 inlet and n outlets, for n usually being two.
- itemize
  - attr\_Substance, attr\_Volume\_Substance, attr\_Max\_In\_Flow\_1, attr\_Max\_In\_Flow\_2, attr\_Curr\_In\_Flow\_1, attr\_Curr\_In\_Flow\_2, attr\_Max\_Out\_Flow, attr\_Curr\_Out\_Flow.
- Possibly other attributes.

**fork:** A fork has one inlet and n outlets, for n usually being two.
- itemize
- Possibly other attributes.

**treat:** Besides the attributes of the join and fork units, the treat units are characterised by the operations that they can perform on their conjoined materials.
- Operations: Usually a treatment unit can perform one operation on its ‘embodied’ (conjoined) materials. But it is always good to generalise, so we say there are n ≥ 1 operations. Each operation is characterised by a “recipe scaled” signature. m is the number of distinct materials, each of substance Substance\_i, f\_i is an appropriate fraction, 0 ≤ f\_i ≤ 1.
  - \( o_1: \text{Operation}_1: f_1 \times \text{Substance}_1 \times f_2 \times \text{Substance}_2 \times \cdots \times f_m \times \text{Substance}_m \)
  - \( o_2: \text{Operation}_2: f_2 \times \text{Substance}_1 \times f_2 \times \text{Substance}_2 \times \cdots \times f_m \times \text{Substance}_m \)
  - \( \cdots \)
  - \( o_m: \text{Operation}_n: f_m \times \text{Substance}_1 \times f_m \times \text{Substance}_2 \times \cdots \times f_m \times \text{Substance}_m \)
- Possibly other attributes.

**dispose:** A disposal conjoin usually has indefinite capacity (i.e., volume).
- attr\_Substance, attr\_Volume\_Substance, attr\_Max\_In\_Flow, attr\_Curr\_In\_Flow.
- Possibly other attributes.

We could likewise consider material-parts conjoins, but leave that to the reader.

4.4.9 Fuzzy Attributes

Fuzzy sets introduced, notably, by Lotfi Zadeh [379, 1965]\(^{39}\) are somewhat like sets whose elements have degrees of membership. Fuzzy set is a mathematical model of vague qualitative or quantitative data, frequently generated by means of the natural language. We shall thus distinguish between fuzzy attribute

---

\(^{39}\) – and, it appears, also, same year, by Dieter Klaua, [260, 261].
values, i.e., vague qualitative values, and fuzzy attributes, i.e., vague quantitative types. Before Klaua and Zadeh fuzziness in logic had been studied as infinite-valued logic Łukasiewicz\textsuperscript{40} and Alfred Tarski\textsuperscript{41}.

4.4.9.0.1 Fuzzy Sets and Fuzzy Logic

We shall informally characterise fuzziness. In classical set theory an element is either a member of some set or it is not, i.e., true or false. In fuzzy set theory an element has a degree, indicated, for example, by a real number in the interval from and including 0 to and including 1. If membership degree is 0 the element is not in the set. If membership degree is 1 the element is certainly in the set. So when we speak of a fuzzy element, as being either of an attribute or an attribute value, then we should indicate its “membership degree”. For the logic of reasoning over fuzzy attribute values and fuzzy attributes we refer to classical textbooks on fuzzy logic and fuzzy sets, e.g., [254].

4.4.9.0.2 Fuzzy Attribute Types

So we can think of an attribute \(A\) as being fuzzy, is\_fuzzy(\(A\)) to mean that its values are fuzzy, i.e., lie in the open interval from and including 0 to and including 1.

4.4.9.0.3 Fuzzy Attribute Values

And these values can be represented, in RSL, by \textbf{Reals}:

- type \(A\): fuzzy: Real

4.4.9.0.4 Fuzzy Reasoning

A las, we shall not, in this monograph, explore the possibilities of modelling domains using Fuzzy Logic!

4.4.9.0.5 Fuzziness: A Possible Research Topic?

Instead we urge readers to do so. The research field of fuzzy sets, logic, systems and engineering is very large. We refer to such peer reviewed journals as

- Fuzzy Sets and Systems. Elsevier journals.elsevier.com/fuzzy-sets-and-systems;
- International Journal of Fuzzy Logic and Intelligent Systems. ijfis.org/main.html;
- Journal of Intelligent & Fuzzy Systems, content.iospress.com/journals/journal-of-intelligent--and-fuzzy-systems/P.

4.5 Intentionality

The conjoin concept, that of relating some endurants more strongly, in the form of conjoins, reflects one or more intentions. In the next section we shall encircle the ‘intention’ concept by quoting from Kai Sorlander’s Philosophy [345, 346, 347, 348].


4.5 Intentionality

4.5.1 Issues Leading Up to Intentionality

Causality of Purpose: If there is to be the possibility of language and meaning then there must exist primary entities which are not entirely encapsulated within the physical conditions; that they are stable and can influence one another. This is only possible if such primary entities are subject to a supplementary causality directed at the future: a causality of purpose.

Living Species: These primary entities are here called living species. What can be deduced about them? They are characterised by causality of purpose: they have some form they can be developed to reach; and which they must be causally determined to maintain; this development and maintenance must occur in an exchange of matter with an environment. It must be possible that living species occur in one of two forms: one form which is characterised by development, form and exchange, and another form which, additionally, can be characterised by the ability to purposeful movements. The first we call plants, the second we call animals.

Animate Entities: For an animal to purposefully move around there must be “additional conditions” for such self-movements to be in accordance with the principle of causality: they must have sensory organs sensing among others the immediate purpose of its movement; they must have means of motion so that it can move; and they must have instincts, incentives and feelings as causal conditions that what it senses can drive it to movements. And all of this in accordance with the laws of physics.

Animals: To possess these three kinds of “additional conditions”, must be built from special units which have an inner relation to their function as a whole; Their purposefulness must be built into their physical building units, that is, as we can now say, their genomes. That is, animals are built from genomes which give them the inner determination to such building blocks for instincts, incentives and feelings. Similar kinds of deduction can be carried out with respect to plants. Transcendently one can deduce basic principles of evolution but not its details.

Humans – Consciousness and Learning: The existence of animals is a necessary condition for there being language and meaning in any world. That there can be language means that animals are capable of developing language. And this must presuppose that animals can learn from their experience. To learn implies that animals can feel pleasure and distaste and can learn. One can therefore deduce that animals must possess such building blocks whose inner determination is a basis for learning and consciousness.

Language: Animals with higher social interaction uses signs, eventually developing a language. These languages adhere to the same system of defined concepts which are a prerequisite for any description of any world: namely the system that philosophy lays bare from a basis of transcendental deductions and the principle of contradiction and its implicit meaning theory. A human is an animal which has a language.

Knowledge: Humans must be conscious of having knowledge of its concrete situation, and as such that human can have knowledge about what he feels and eventually that human can know whether what he feels is true or false. Consequently a human can describe his situation correctly.

Responsibility: In this way one can deduce that humans can thus have memory and hence can have responsibility, be responsible. Further deductions lead us into ethics.

We shall not further develop the theme of living species: plants and animals, thus excluding, most notably humans, in this chapter. We claim that the present chapter, due to its foundation in Kai Sierlander’s Philosophy, provides a firm foundation within which we, or others, can further develop this theme: analysis & description of living species.

Intentionality: Intentionality as a philosophical concept is defined by the Stanford Encyclopedia of Philosophy as “the power of minds to be about, to represent, or to stand for, things, properties and states of affairs.”

Intentional Pull: Two or more artefactual parts of different sorts, but with overlapping sets of intents may exert an intentional “pull” on one another. This intentional “pull” may take many forms. Let \( p_X : X \) and \( p_Y : Y \) be two parts of different sorts \((X,Y)\), and with common intent, i.e., Manifestations of these, their common intent must somehow be subject to constraints, and these must be expressed predicatively.

When a composite or conjoin artefact models “itself” as put together with a number of other endurants then it does have an intentionality and the components’ individual intentionalities does, i.e., shall relate to that. The composite road transport system has intentionality of the road serving the automobile part, and the automobiles have the intent of being served by the roads, across “a divide”, and vice versa, the roads of serving the automobiles.

Internal Qualities

Natural endurants, for example, rivers, lakes, seas and oceans become, in a way, artefacts with and when mankind using them for transport; natural gas becomes an artefact when drilled for, exploited and piped; and harbours make no sense without artefactual boats sailing on the natural water.

This, perhaps vague, concept of intentionality has yet to be developed into something of a theory. Despite that this is yet to be done, cf. Exercise 12 on Page 119, we shall proceed to define an intentionality analysis function. First we postulate a set of intent designators. An intent designator is really a further undefined quantity. But let us, for the moment, think of them as simple character strings, that is, literals, for example "road", "hub", "link", "automobile", "transport", etc.

Analysis Function Prompt 19 analyse_intentionality:

The domain analyser analyses an endurant as to the a finite number of intents, zero or more, with which the analyser judges the endurant can be associated. The method provides the domain analysis prompt:

• analyse_intentionality directs the domain analyser to observe a set of intents.
  value analyse_intentionality(e) = \{i_1, i_2, ..., i_n\} ⊆ Intent

Example 65 We illustrate the concept of intentional "pull":

222 automobiles include the intent of 'transport',
223 and so do hubs and links.

222 analyse_intentionality: A → ("transport"...) set
223 analyse_intentionality: H → ("transport"...) set
223 analyse_intentionality: L → ("transport"...) set

Manifestations of "transport" is reflected in automobiles having the automobile position attribute, APos, Item 181 Pg. 100, hubs having the hub traffic attribute, H_traffic Item 169 Pg. 98, and in links having the link traffic attribute, L_traffic Item 173 Pg. 99.

224 Seen from the point of view of an automobile there is its own traffic history, A_hist, which is a (time ordered) sequence of timed automobile’s positions;
225 seen from the point of view of a hub there is its own traffic history, H_traffic Item 169 Pg. 98, which is a (time ordered) sequence of timed maps from automobile identities into automobile positions; and
226 seen from the point of view of a link there is its own traffic history, L_traffic Item 173 Pg. 99, which is a (time ordered) sequence of timed maps from automobile identities into automobile positions.

The intentional "pull" of these manifestations is this:

227 The union, i.e. proper merge of all automobile traffic histories, AllATH, must now be identical to the same proper merge of all hub, AllHTH, and all link traffic histories, AllLTH.

\[ \text{Type} \]

224 A\_Hi = (T × APos)*
169 H\_Trf = A\_UI ⇒ (TIME × APos)*
173 L\_Trf = A\_UI ⇒ (TIME × APos)*
227 AllATH = TIME ⇒ (A∪I ⇒ APos)
227 AllHTH = TIME ⇒ (A∪I ⇒ APos)
227 AllLTH = TIME ⇒ (A∪I ⇒ APos)

\[ \text{Axiom} \]

227 let allA = mrg\_AllATH((\{attr\_A\_Hi(a)|a:A\_a ∈ as\}),
227 allH = mrg\_AllHTH((\{attr\_H\_Trf(h)|h:H\_h ∈ hs\}),
43 Seas are smaller than oceans and are usually located where the land and ocean meet. Typically, seas are partially enclosed by land. The Sargasso Sea is an exception. It is defined only by ocean currents [oceanservice.noaa.gov/facts/oceanorsea.html].
We leave the definition of the four merge functions to the reader!

Discussion: We endow each automobile with its history of timed positions and each hub and link with their histories of timed automobile positions. These histories are facts! They are not something that is laboriously recorded, where such recordings may be imprecise or cumbersome. The facts are there, so we can (but may not necessarily) talk about these histories as facts. It is in that sense that the purpose (‘transport’) for which man let automobiles, hubs and link be made with their ‘transport’ intent are subject to an intentional “pull”. It can be no other way: if automobiles “record” their history, then hubs and links must together “record” identically the same history!

Please note, that intents are not [thought of as] attributes. We consider intents to be a fourth, a comprehensive internal quality of endurants. They, so to speak, govern relations between the three other internal quality of endurants: the unique identifiers, the mereologies and the attributes. That is, they predicate them, “arrange” their comprehensiveness. Much more should be said about intentionality. It is a truly, I believe, worthy research topic of its own. We refer to Exercise 12 on Page 119.

Example 66 Let us illustrate the issues “at play” here.

• Consider a road transport system uod.
  ∞ Applying analyse_intentionality(uod) may yield the set {"transport", ...}.

• Consider a financial service industry, fss.
  ∞ Applying analyse_intentionality(fss) may yield the set {"interest on deposit", ...

• Consider a health care system, hcs.
  ∞ Applying analyse_intentionality(hcs) may yield the set {"cure diseases", ...}.

What these analyses of intentionality yields, with respect to expressing intentional pull, is entirely of the discretion of the domain analyser & describer.

An Aspect of Comprehensiveness of Internal Qualities

We bring the above example, Example 66, to indicate, as the name of the example reveals, “An Aspect of Comprehensiveness of Internal Qualities”. That the various components of artefactual systems relate in – further to be explored – ways. In this respect, performing domain analysis & description is not only an engineering pursuit, but also one of research. We leave it to the readers to pursue this research aspect of domain analysis & description – while referring to Exercise 12 on Page 119.

4.5.2 Artefacts

Humans create artefacts – for a reason, to serve a purpose, that is, with intent. Artefacts are like parts. They satisfy the laws of physics – and serve a purpose, fulfill an intent.

4.5.3 Assignment of Attributes

So what can we deduce from the above, a little more than two pages?

The attributes of natural parts and natural materials are generally of such concrete types – expressible as some real with a dimension of the International System of Units: https://physics.nist.gov/cuu/Units/units.html. Attribute values usually enter differential equations and integrals, that is, classical calculus.

44 or thought technologically in-feasible – at least some decades ago!
45 Basic units are meter, kilogram, second, Ampere, Kelvin, mole, and candela. Some derived units are: Newton: kg×m×s⁻², Weber: kg×m²×s⁻²×A⁻¹, etc.
The attributes of humans, besides those of parts, significantly includes one of a usually non-empty set of intents. In directing the creation of artefacts humans create these with an intent.

**Intentional Pull, II**

**Example 67** These are examples of human intents: they create roads and automobiles with the intent of transport, they create houses with the intents of living, offices, production, etc., and they create pipelines with the intent of oil or gas transport.

Human attribute values usually enter into modal logic expressions.

### 4.5.3.1 Artefacts, including Man-made Materials:

Artefacts, besides those of parts, significantly includes a usually singleton set of intents.

**Example 68** Roads and automobiles possess the intent of transport; houses possess either one of the intents of living, offices, production; and pipelines possess the intent of oil or gas transport.

Artefact attribute values usually enter into mathematical logic expressions.

We leave it to the reader to formulate attribute assignment principles for plants and non-human animals.

### 4.5.4 Galois Connections

Galois Theory was first developed by Évariste Galois [1811-1832] around 1830. Galois theory emphasizes a notion of Galois connections. We refer to standard textbooks on Galois Theory, e.g., [351, 2009].

#### 4.5.4.1 Galois Theory: An Ultra-brief Characterisation

To us, an essence of Galois connections can be illustrated as follows:

- Let us observe properties of a number of endurants, say in the form of attribute types.
- Let the function \( F \) map sets of entities to the set of common attributes.
- Let the function \( G \) map sets of attributes to sets of entities that all have these attributes.
- \((F, G)\) is a Galois connection
  - if, when including more entities, the common attributes remain the same or fewer, and
  - if when including more attributes, the set of entities remain the same or fewer.
- \((F, G)\) is monotonously decreasing.

**Example 69**

- We have a collection of LEGO™ blocks.
- From this collection, \( A \), we identify the red square blocks, \( e \).
- That is \( F(A) = \{ \text{attr\_Color}(e) = \text{red}, \text{attr\_Form}(e) = \text{square} \} \).
- We now add all the blue square blocks.
- And obtain \( A' \).

---

46 en.wikipedia.org/wiki/Galois_theory

47 The following is an edited version of an explanation kindly provided by Asger Eir, e-mail, June 5, 2020 [142, 143, 82].
• Now the common properties are their squareness: $F(A')$ is $B' = \{\text{attr_Form(e)=square}\}$.
• More blocks as argument to $F$ yields fewer or the same number of properties.
• The more entities we observe, the fewer common attributes they possess.

### Civil Engineering: Consultants and Contractors

**Example 70** Less playful, perhaps more seriously, and certainly more relevant to our endeavour, is this next example.

- Let $X$ be the set of civil engineering, i.e., building, consultants, i.e., those who, like architects and structural engineers design buildings – of whatever kind.
- Let $Y$ be the set of building contractors, i.e., those firms who actually implement, i.e., build to, those designs.
- Now a subset, $X_{bridges}$ of $X$, contain exactly those consultants who specialise in the design of bridges, with a subset, $Y_{bridges}$, of $Y$ capable of building bridges.
- If we change to a subset, $X'' = X_{bridges,tunnels}$ of $X$, allowing the design of both bridges and tunnels, then we obtain a corresponding subset, $Y_{bridges,tunnels}$, of $Y$.
- So when
  - we enlarge the number of properties from ‘bridges’ to ‘bridges and tunnels’,
  - we reduce, most likely, the number of contractors able to fulfill such properties,
  - and vice versa,
- then we have a Galois Connection.

### 4.5.4.2 Galois Connections and Intentionality

We have a hunch\footnote{Hunch: a feeling or guess based on intuition rather than fact.}! Namely that there are some sort of Galois Connections with respect to intentionality.

### 4.5.4.3 Galois Connections and Intentionality: A Possible Research Topic?

We leave to to the interested reader to pursue this line of inquiry.

### 4.6 Systems Modelling

#### 4.6.1 General

In Sect. 4.3.7, as well as in numerous examples we started to reveal some “classes” of domains for the modelling of which it appears that there are some “standard” techniques. For general, usually bi-directed networks of usually atomic parts, we analysed & described these graphs into sets of units whose mereology “revealed” their “interconnection”. For less general, usually directed, acyclic networks of usually conjoins, we analysed & described these graphs also into sets of units, now the conjoins, whose mereology now “revealed” their “interconnection”. For less topologically, more conceptually and intentionally related aggregations\footnote{for lack of a better term} of endurants, we analysed & described as possibly hierarchically organised composite endurants, whose mereology, in their way, “revealed” the “interconnection” of the aggregations.

#### 4.6.2 Passively Mobile Endurants

Some endurants are mobile. Mobile endurants are either actively mobile, i.e., move on their own accord, or passively mobile, i.e., are transported by other endurants. Usually passively mobile endurants are expressed
as siblings of part-parts conjoins – where the ‘parts’ usually consist of a definite number of these: usually zero, one or two.

### Credit Card Shopping System

**Example 71** A credit card shopping system\(^{51}\) consists of (i) credit card (user)s, (ii) shops and (iii) credit card honoring banks. The shops offer for sale and users hoard merchandise. We suggest to model, as a fourth element of the system, (iv) the merchandise. And let credit card user and shop attributes reflect their merchandise by their unique identifiers.

### Container Terminal Port

**Example 72** A container terminal port\(^{52}\) consists of (i) vessels, (ii) vessel to/from quay cranes, (iii) quay crane to/from stack trucks, (iv) stack or land truck to/from stack cranes, (v) stacks, and (vi) land trucks. Vessels and stacks hold any number of containers over indefinite time intervals. Cranes and trucks hold zero, one or two containers over expectedly short time intervals. We suggest to model, as a seventh element, of a container terminal port (vii) containers; then let vessel, crane, truck and stack attributes reflect their zero or more containers by their unique identifiers.

### General Hospital System

**Example 73** A general hospital consists of (i) beds, (ii) staff, and (iii) patients. Patients occupy beds are operated upon by medical doctor staff, are otherwise cared for by nurse staff, et cetera. We suggest to model occupants of beds, patients on operating tables, and patients being cared for by nurses by unique patient identifiers.

Thus we suggest the following modelling choices: Actively mobile endurants shall be transcendentally deduced into behaviours. Passively mobile endurants if "embodied" by actively mobile endurants, shall not. If, instead, these passively mobile endurants are modelled as a major set of endurants of their domain they will be modelled as behaviours – with few other actions than responding to “who, at the moment, transports them”. All this should be clear after Chapter 6.

### 4.7 Discussion of Endurants

Domain descriptions are, as we have already shown, formulated, both informally and formally, by means of abstract types, that is, by sorts for which no concrete models are usually given. Sorts are made to denote possibly empty, possibly infinite, rarely singleton, sets of entities on the basis of the qualities defined for these sorts, whether external or internal. By junk we shall understand that the domain description unintentionally denotes undesired entities. By confusion we shall understand that the domain description unintentionally have two or more identifications of the same entity or type. The question is can we formulate a [formal] domain description such that it does not denote junk or confusion? The short answer to this is no! So, since one naturally wishes “no junk, no confusion” what does one do? The answer to that is one proceeds with great care!

### 4.8 A Domain Discovery Process, II

We shall again emphasize some aspects of the domain analyser & describer method. A method principle is that of exhaustively analyse & describe all internal qualities of the domain under scrutiny. A method

\(^{51}\) See imm.dtu.dk/~dibj/2016/credit/accs.pdf
\(^{52}\) See imm.dtu.dk/~dibj/2018/yangshan/maersk-pa.pdf
technique implied here is that sketched below. The method tools are here all the analysis and description prompts covered so far.

The predecessor of this section is Sect. 3.20 on Page 74. Please be reminded of Discovery Schema 0’s declaration of Notice Board variables (Page 74). In this section we collect (i) the description of unique identifiers of all parts of the state; (ii) the description of mereologies of all parts of the state; and (iii) the description of attributes of all parts of the state. (iii) We finally gather these into the discover internal endurant qualities procedures.

An Endurant Internal Qualities Domain Analysis and Description Process

| discover_uids: Unit → Unit |
| discover_uids(): ≡ |
| for ∨ v ∈ gen |
| do txt := txt † [type_name(v)→txt(type_name(v))]^([describe_unique_identifier(v)]) | end |
| discover_mereologies: Unit → Unit |
| discover_mereologies(): ≡ |
| for ∨ v ∈ gen |
| do txt := txt † [type_name(v)→txt(type_name(v))]^([describe_mereology(v)]) | end |
| discover_attributes: Unit → Unit |
| discover_attributes(): ≡ |
| for ∨ v ∈ gen |
| do txt := txt † [type_name(v)→txt(type_name(v))]^([describe_attributes(v)]) | end |

discover internal endurant qualities: Unit → Unit |
| discover internal endurant qualities(): ≡ |
| discover_uids(); |
| axiom [ all parts have unique identifiers ] |
| discover_mereologies(); |
| axiom [ all unique identifiers are mentioned in sum total of ] |
| [ all mereologies and no isolated proper sets of parts ] |
| discover_attributes(); |
| axiom [ sum total of all attributes span all parts of the state ] |

We shall comment on the axioms in the next section.

4.9 Domain Description Laws

The axioms of the immediately above Discovery Schema expresses some domain facts: [i] the uniqueness of part identifiers; [ii] that mereologies mention all parts and that the mereologies of no proper subset of parts subset of parts refer only to parts of that subset; and [iii] that part attributes, when they refer, refer only to parts of the state.

4.10 Summary

This chapter’s main title was: DOMAINS – Towards a Statics Ontology. The term ‘statics’ pertain to qualities of the ‘Domain’, not to its ‘Ontology’. So, an aspect of the ontology of a domain, such as we have studied it and such as we ordain one aspect of domain analysis & description, is about somehow measurable properties, or about historical actions and events to which endurants of the domain have been subjected, that is, the internal qualities. For that study & practice we have suggested a number of analysis & description prompts.

4.10.1 The Description Schemas

We have culminated this chapter with description prompts for unique identifier description schema 9 on Page 86, mereology description schema 10 on Page 90 and attribute description schema 11 on Page 95.
They all describe in, in our case RSL, the domain description text to ‘produce’ when internal quality analysing & describing a given endurant; but what about the description of those endurants revealed by the analysis & description of that given endurant?

The answer is simple. That is up to you! The domain analysis & description method gives you the tools, some techniques and a few principles. But:

- A principle of the method could be to secure that all relevant, i.e., implied, endurants are analysed & described.
- A technique could be to, somehow, “set aside” all those endurants revealed by the analysis & description of any given endurant – with the proviso that no endurant, of type, for example, $P$, is analysed & described more than once.

Same answer that we gave in Sect. 3.22.1 Page 77. A technique, such as alluded to above, is show ‘formalised’ in the pseudo-program of Sect. 4.8 on Page 116.

4.10.2 Modelling Choices

In this chapter we have put forward some advice on description choices: We refer to Modelling Choices 5 on Page 91 and 6 on Page 96. The analysis predicates and functions are merely aids. They do not effect descriptions, but descriptions are based on the result of their inquiry. Real decisions are made when effecting a description function. So the rôle of these modelling choice paragraphs is to alert the describer to make judicious choices.

4.10.3 Method Principles, Techniques and Tools

Recall that by a method we shall understand a set of principles for selecting and applying a set of techniques using a set of tools in order to construct an artefact.

4.10.3.1 Principles of Internal Qualities

In this chapter we have illustrated the use of the following techniques:

**Divide and Conquer:** Application of this principle has in this chapter been quite pronounced: the ‘divisions’ are those of first (i) the analysis & description of unique identification, then (ii) the analysis & description of mereologies, and then, finally, (iii) the analysis & description of attributes – and in that order. We have found, in numerous case studies [80], that any other “strict” order very often brings confusion.

**Representational Abstract:** Application of this principle has, in this chapter, been to the type definitions of unique identifiers, mereologies and attributes. For unique identifiers in that no representation need be prescribed. For mereologies in that all we are really interested in are which parts “partake” in part-to-part relations. For attributes we have not directed the domain analyser cum describer as how express possible attribute type expressions, and often we just identify an attribute type by its identifier.

4.10.3.2 Techniques of Internal Qualities

In this chapter we have illustrated the use of the following principles:

**Invariants:** We remind the reader of Item 49 on Page 8, and refer to Example 45 on Page 89: Uniqueness of Road Net Identifiers, Example 48 on Page 91: Invariance of Road Nets and Example 58 on Page 98: Invariance of Road Net Traffic States.

**Intentional Pull:** We remind the reader of Item 48 on Page 8, and refer to Example 65 on Page 112: Intentional Pull, I and Example 67 on Page 114: Intentional Pull, II.

53 “Eager-beaver”, inventive “whiz kids” are often caught up in their creativeness and muddles matters up, forgets careful and necessary analyses whose absence often shows up late, and much analysis & description work has to be redone!
**4.10.3.3 Tools**

4.10.3.3.1 **Summary of The Internal Qualities Analysis Calculus**

- `analyse_attribute_types` and `is_physical_attribute`. Page 95
- `is_physical_attribute`. Page 107

4.10.3.3.2 **Summary of The Internal Qualities Description Calculus**

- `describe_unique_identifier`, Page 86
- `describe_mereology` and `describe_attributes`. Page 95

**4.11 Bibliographical Notes**

We refer to [70, Sect. 5.3] for a thorough, 2016–2017, five page review of types in formal specification and programming languages.

**4.12 Exercise Problems**

**4.12.1 Research Problems**

**Exercise 11** A Research Challenge. **Fuzzy Descriptions**: Experiment with, present examples, and, possibly, develop analysis & description prompts for Fuzzy attributes.

**Exercise 12** A Research Challenge. **Intentionality**: Suggest possible intentions and possible intentional pulls for domains of artefacts, say as they are mentioned in the Term Projects section. Present possible examples. More generally, develop a theory of intentionality.

**Exercise 13** A Research Challenge. **Galois Connections**: Study Galois Connections as, for example presented in [158, Ganter & Wille]. Then search for such connections with respect to internal qualities of pairs of different sort discrete endurants. Present examples. Suggest possible [?] analysis & description prompts.

**Exercise 14** A PhD Student Problem. **Living Species: Humans**: Suggest an outline mereology and attribute concepts for humans.

**Exercise 15** A PhD Student Problem. **Michael Jackson’s Categories of Attributes**: Suggest a critique of Jackson’s categories of attributes, cf. Sect. 4.4.2.3.

- Is Jackson’s categorisation equally applicable to natural parts as well as to discrete artefacts?
- Somehow or other, do discrete artefacts mandate a different categorisation?
- Suggest a categorisation for discrete artefacts.
4.12.2 Student Exercises

Exercise 16 An MSc Student Exercise. Unique Document Identification: We refer to Exercise 3 on Page 80.

[Q1] you are to narrate and formalise unique identification for persons and documents. We refer to upcoming Exercises 17, 18 and Exercise 29 on Page 160.

Exercise 17 An MSc Student Exercise. Document System Mereologies: We refer to Exercises 3 on Page 80 and 16.

We anticipate and elaborate on the actions that Exercise 29 on Page 160 will be handling.

create: The thus created document shall record (i) the identity of its [person] creator, (ii) time of creation, and (iii) the text it now contains.

edit: The thus edited document shall record (i) the identity of its [person] editor, (ii) the time of edit, and (iii) the changes being made to the

- master document, whose text is $\tau_M$,
- being edited into the edited document, whose text is $\tau_E$,
- such that these changes can be "seen" – for example as follows:
  - there is a "forward" editing function, $e_F$,
  - and an "undo" editing function, $e_U$,
  - such that the text now recorded, in the edited document, is $e_F(\tau_M)$,
  - and such that $e_U(e_F(\tau_M)) = \tau_M$.

read: The thus read document shall record (i) the identity of its [person] reader and (ii) the time of read.

copy: As a result of a copy we now have one more document in our system: besides the document, the original, we now also have the copy.

The original document shall record (i) the identity of its [person] who had copied this document, (ii) the identity of the copy, and (iii) the time of copying.

The copy document shall record (i) the identity of its [person] who made this copy, (ii) the identity of the original document (from which it was made), and (iii) the time of copying.

shred: Let the identity of the document being shredded be $d_i$.

Whereas there was a document of that identity, i.e., $d_i$, there is, “officially”, no longer such a document.

But, since persons can talk about the historical existence of $d_i$, we may have to keep track of all shredded documents. See question [Q2] of Exercise 3 on Page 80.

Therefore documents in such a “shredded” document archive must record (i) the identity of the person who did the shredding and (ii) the time of shredding.

[Q1] you are to narrate and formalise mereologies for persons and documents.

We refer to upcoming Exercises 18 and 29 on Page 160.

Exercise 18 An MSc Student Exercise. Document System Attributes: We refer to Exercises 3 on Page 80 and 16.

In addressing question [Q1] below you will have studied the indented text of Exercise 17.

[Q1] you are to narrate and formalise attributes for persons and documents.

We refer to upcoming Exercise 29 on Page 160.

Exercise 19 An MSc Student Exercise. A Simple Consumer–Bank–Retailer Credit Card System: The credit card system involves consumers, retailers banks and credit cards. A credit card is an attribute of consumers, one per consumer. Consumers have bank accounts, one per consumer. Retailers stock merchandise for sale, all merchandise are distinct, and have a price tag. Retailers have bank accounts, one per retailer. Banks hold accounts for consumers and retailers. Credit cards identify their [consumer] holder, one per credit card. Now to the problem to be solved. Please note item [g] before you start writing down your solutions.
You are to formulate [a] appropriate sorts for consumers, retailers and banks as parts; [b] appropriate sets of unique identifiers, mereologies and [c] attributes for consumers, retailers and banks; You are to express appropriate well-formedness conditions for [d] all mereologies and [e] all attributes. You are to express an intentional pull [f] relating the bank balances of consumers and retailers. While doing this, you are to [g] spot what might, inadvertently, have been left out in the above, first paragraphs ‘presentation’ of this, albeit simple consumer-retailer-bank credit card system.

### 4.12.3 Term Projects

We continue the term projects of Sect. 3.24.3 on Page 81.

For the specific domain topic that a group is working on it is to treat, for example, in separate weeks, these topics in the order listed:

- **unique identifiers**, cf. Sect. 4.2,
- **mereology**, cf. Sect. 4.3, and
- **attributes**, cf. Sect. 4.4, the latter possibly over two weeks.

**Exercise 20** An MSc Student Exercise. **The Consumer Market, Internal Qualities**: We refer to Exercise 4 on Page 82. You are, in turn, to analyse and describe

- the unique identifiers,
- mereologies,
- a suitable sample of attributes, and
- possible intentional pulls

of consumer markets.

**Exercise 21** An MSc Student Exercise. **Financial Service Industry, Internal Qualities**: We refer to Exercise 5 on Page 82. You are, in turn, to analyse and describe

- the unique identifiers,
- mereologies,
- a suitable sample of attributes, and
- possible intentional pulls

of financial service industries.

**Exercise 22** An MSc Student Exercise. **Container Line Industry, Internal Qualities**: We refer to Exercise 6 on Page 82. You are, in turn, to analyse and describe

- the unique identifiers,
- mereologies,
- a suitable sample of attributes, and
- possible intentional pulls

of container lines.

**Exercise 23** An MSc Student Exercise. **Railway Systems, Internal Qualities**: We refer to Exercise 7 on Page 82. You are, in turn, to analyse and describe

- the unique identifiers,
- mereologies,
- a suitable sample of attributes, and
- possible intentional pulls

of railway systems.

**Exercise 24** A PhD Student Problem. **Part-Material Conjoins: Canals, Internal Qualities**: We refer to Example 8 on Page 82. You are, in turn, to analyse and describe

- the unique identifiers,
- mereologies,
- a suitable sample of attributes, and
- possible intentional pulls

of canal systems.

**Exercise 25** A PhD Student Problem. **Part-Materials Conjoins: Rum Production, Internal Qualities**: We refer to Exercise 9 on Page 82. You are, in turn, to analyse and describe
Internal Qualities

- the unique identifiers,
- mereologies,
- a suitable sample of attributes, and
- possible intentional pulls

of rum production.

**Exercise 26** A PhD Student Problem. **Part-Materials Conjoins: Waste Management, Internal Qualities**: We refer to Exercise 10 on Page 82. You are, in turn, to analyse and describe

- the unique identifiers,
- mereologies,
- a suitable sample of attributes, and
- possible intentional pulls

of waste management.

These exercise problems are continued in Sects. 6.14.3 on Page 161, 7.11.2 on Page 193 and 8.9.2 on Page 241.
5

TRANSCENDENTAL DEDUCTION

In this chapter we discuss the concept of transcendental deduction.

It should be clear to the reader that in domain analysis & description we are reflecting on a number of philosophical issues; first and foremost on those of ontology. For this chapter we reflect on a sub-field of epistemology, we reflect on issues of transcendental nature. Should you wish to follow-up on the concept of transcendentality, we refer to [191, Immanuel Kant], [242, Oxford Companion to Philosophy, pp 878–880], [12, The Cambridge Dictionary of Philosophy, pp 807–810], [110, The Blackwell Dictionary of Philosophy, pp 54–55 (1998)], [348, Sørlander] and Chapter 1.

**Definition: 61** Transcendental, II: By transcendental we shall understand the philosophical notion: the a priori or intuitive basis of knowledge, independent of experience. A priori knowledge or intuition is central: By a priori we mean that it not only precedes, but also determines rational thought.

**Definition: 62** Transcendental Deduction, II: By a transcendental deduction we shall understand the philosophical notion: a transcendental “conversion” of one kind of knowledge into a seemingly different kind of knowledge.

Some Transcendental Deductions

Example 74 We give some intuitive examples of transcendental deductions. They are from the “domain” of programming languages. There is the syntax of a programming language, and there are the programs that supposedly adhere to this syntax. Given that, the following are now transcendental deductions. The software tool, a syntax checker, that takes a program and checks whether it satisfies the syntax, including the statically decidable context conditions, i.e., the static semantics – that tool is one of several forms of transcendental deductions.

The software tools, an automatic theorem prover and a model checker, for example SPJ [241], that takes a program and some theorem, respectively a Promela statement, and proves, respectively checks, the program correct with respect the theorem, or the statement.

A compiler and an interpreter for any programming language.

Yes, indeed, any abstract interpretation [129, 96] reflects a transcendental deduction: firstly, these examples show that there are many transcendental deductions; secondly, they show that there is no single-most preferred transcendental deduction.

A transcendental deduction, crudely speaking, is just any abstraction that can be “linked” to another, not by logical necessity, but by logical (and philosophical) possibility.

**Definition: 63** Transcendentality: By transcendentality we shall here mean the philosophical notion: the state or condition of being transcendental.

Transcendentality

---

1 ACL2 [258], Coq [28], Isabelle/HOL [299], STeP [93], PVS [303] and Z3 [94]
Example 75  We can speak of a bus in at least three senses:

(i) The bus as it is being "maintained, serviced, refueled";
(ii) the bus as it "speeds" down its route; and
(iii) the bus as it "appears" (listed) in a bus time table.

The three senses are:

(i) as an endurant (here a part),
(ii) as a perdurant (as we shall see, a behaviour), and
(iii) as an attribute.  

The above example, we claim, reflects transcendentality as follows:

(i) We have knowledge of an endurant (i.e., a part) being an endurant.
(ii) We are then to assume that the perdurant referred to in (ii) is an aspect of the endurant mentioned in (i) – where perdurants are to be assumed to represent a different kind of knowledge.
(iii) And, finally, we are to further assume that the attribute mentioned in (iii) is somehow related to both (i) and (ii) – where at least this attribute is to be assumed to represent yet a different kind of knowledge.

In other words: two (i–ii) kinds of different knowledge; that they relate must indeed be based on a priori knowledge. Someone claims that they relate! The two statements (i–ii) are claimed to relate transcendentally.  

---

2 – in this case rather: as a fragment of a bus time table attribute.
3 – the attribute statement was “thrown” in “for good measure”, i.e., to highlight the issue!
In this chapter we transcendentally “morph” parts into behaviours. We analyse that notion and its constituent notions of actors, channels and communication, actions and events.

The main transcendental deduction of this chapter is that of associating with each part a behaviour. This section shows the details of that association. Perdurants are understood in terms of a notion of state and a notion of time.

6.1 Structure of this Chapter

In order to culminate, in Sect. 6.7 we need to treat a number of pre-requisite topics. There are quite a few of these, so a summary-of-what-is-to-come seems reasonable.

- Section 6.2 covers primarily the notion of domain states in the form of CSP variables – one for each of the parts having monitorable attributes;
- Sect. 6.3 surveys the notions of actors, actions, events and behaviours;
- Sect. 6.4 discuss the modelling of concurrent domain behaviours in terms of CSP processes – with brief subsections on CSP and Petri nets;
- Sect. 6.5 then introduces the notions of CSP channels, output and input – to model interaction between domain behaviours;
- Sect. 6.6 discusses action, event and behaviour signatures in general;
- Sect. 6.7 is now ready to tackle the important issue of defining domain behaviours, including their signatures;
- Sect. 6.8 shows how to express the initialisation of a running domain behaviour;
- Sect. 6.10 loosely discusses the modelling of domain actions; while
- Sect. 6.11 briefly touches upon the modelling of domain events.

Finally Sect. 6.12 follows up on the domain discovery process of Sects. 3.20 and 4.8.

Other sections provide elucidation or summary observations.

6.2 States and Time

We first covered the notions of state in Sects. 1.3.2.7 on Page 14 and 3.15 on Page 61 and time in Sect. 2.5 on Page 24.

6.2.1 The Issue of States

Example 33 on Page 62 illustrated the idea of expressing the values of all parts having dynamic attributes. We refer to [176] and Appendix Sect. D.6.2 on Page 321.

RSL variables of the form:

\[
\text{variable parts[uid}_P(p)];P := p
\]

\(^0\) The ‘Dynamics’ refer back to ‘DOMAINS’ – not to ‘Ontology’!
are to be declared to model parts that have monitorable attributes; informally:

\[
\text{has\_monitorable\_attributes: } P 
\rightarrow \text{Bool} \\
\text{has\_monitorable\_attributes}(p) \equiv \\
\exists A \cdot A \in \text{analyse\_attributes\_types}(p) \cdot \text{is\_monitorable}(\text{attr\_A}(p))
\]

\[
\text{possible\_variable\_declaration: } P 
\rightarrow \text{RSL-Text} \\
\text{possible\_variable\_declaration}(p) \equiv \\
\text{if has\_monitorable\_attributes}(p) \text{ then } \text{"variable } p[\text{uid}_P(p)] : P := p \text{ " end}
\]

analyse\_attribute\_types is defined in domain analysis function prompt 18 on Page 95.

---

**Translation Schema 1**

When we have ‘collected’ all external endurant descriptions

228 we can, for any given endurant, e, typically a *universe of discourse* domain,
229 calculate all relevant *monitorable variable* declarations;
230 that is, for those parts, p,
231 that have monitorable-only attributes.

229. \text{declaring\_all\_monitorable\_variables: } E 
\rightarrow \text{RSL-Text} \\
229. \text{declaring\_all\_monitorable\_variables}(e) \equiv \\
230. \text{let } ps = \text{calc\_parts}(e) \text{ in} \\
230. \text{for } \forall p \in ps \text{ do possible\_variable\_declaration}(p) \\
229. \text{end end}

---

**State Values versus State Variables**

**Example 76**

Item 116 on Page 62 expresses the *value* of all parts of a road transport system:

116. \text{ps:\(\{UoB|H|L|BC|B|A\}\)-set} \equiv \text{rts}\cup\text{hls}\cup\text{bcs}\cup\text{bs}\cup\text{as}.

232 We now introduce the set of variables, one for each part value of the domain being modelled.

232. \{ \text{variable } vp:\(\{UoB|H|L|BC|B|A\} \mid vp:\(\{UoB|H|L|BC|B|A\} \cdot vp\in ps \} \}

---

**6.2.2 Time Considerations**

We shall, without loss of generality, assume that actions and events are atomic and that behaviours are composite. Atomic perdurants may “occur” during some time interval, but we omit consideration of and concern for what actually goes on during such an interval. Composite perdurants can be analysed into “constituent” actions, events and “sub-behaviours”. We shall also omit consideration of temporal properties of behaviours. Instead we shall refer to two seminal monographs: *Specifying Systems* [263, Leslie Lamport] and *Duration Calculus: A Formal Approach to Real-Time Systems* [380, Zhou ChaoChen and Michael Reichhardt Hansen] (and [41, Chapter 15]). For a seminal book on “time in computing” we refer to the eclectic [156, Mandrioli et al., 2012]. And for seminal book on time at the epistemology level we refer to [353, J. van Benthem, 1991].
6.3 Actors, Actions, Events and Behaviours: A Preview

To us perdurants are further, pragmatically, analysed into actions, events, and behaviours. We shall define these terms below. Common to all of them is that they potentially change a state. Actions and events are here considered atomic perdurants. For behaviours we distinguish between discrete and continuous behaviours.

### 6.3.1 Actors

**Definition:** 64 **Actor:** By an *actor* we shall understand something that is capable of *initiating* and *carrying out* actions, events and behaviours.

The notion of “carrying out” will be made clear in this overall chapter. We shall, in principle, associate an actor with each part\(^1\). These actors will be described as behaviours. These behaviours evolve around a state. The state is the set of qualities, in particular the dynamic attributes, of the associated parts and/or any possible components or materials of the parts.

### 6.3.2 Discrete Actions

**Definition:** 65 **Discrete Action:** By a *discrete action* [366, Wilson and Shpall] we shall understand a foreseeable thing which deliberately and potentially changes a well-formed state, in one step, usually into another, still well-formed state, and for which an actor can be made responsible.

An action is what happens when a function invocation changes, or potentially changes a state.

### 6.3.3 Discrete Events

**Definition:** 66 **Event:** By an *event* we shall understand some unforeseen thing, that is, some ‘not-planned-for’ “action”, one which surreptitiously, non-deterministically changes a well-formed state into another, but usually not a well-formed state, and for which no particular domain actor can be made responsible.

Events can be characterised by a pair of (before and after) states, a predicate over these and, optionally, a time or time interval.

We shall use the RSL concepts of *clauses*, i.e., *expressions* and *statements* to model actions. We shall use the CSP concepts of *channels* and *channel communication*, i.e., *message output*: \( \text{ch}[\ldots]!e \) and *message input*: \( \text{ch}[\ldots]? \) to model events. The notion of event continues to puzzle philosophers [141, 327, 285, 136, 192, 13, 114, 315, 113].

### 6.3.4 Discrete Behaviours

**Definition:** 67 **Discrete Behaviour:** By a *discrete behaviour* we shall understand a set of sequences of potentially interacting sets of discrete actions, events and behaviours.

Discrete behaviours now become the focal point of our investigation. To every part we associate, by transcendental deduction, a behaviour. We shall express these behaviours as CSP processes [238]. For those behaviours we must therefore establish their means of communication via channels; their signatures; and their definitions – as translated from endurant parts.

---

1. This is an example of a transcendental deduction.
The idea is that those are the parts for which we shall define behaviours. That figure, however, and in contrast to Fig. 3.4 on Page 69, shows the composite parts as not containing their atomic parts, but as if they were “free-standing, atomic” parts. That shall visualise the transcendental interpretation as atomic part behaviours not being somehow embedded in composite behaviours, but operating concurrently, in parallel.

6.3.5 Continuous Behaviours

By a continuous behaviour we shall understand a continuous time sequence of state changes. We shall not go into what may cause these state changes. And we shall not go into continuous behaviours in this monograph.

6.4 Modelling Concurrent Behaviours

We choose to exploit the CSP [238] subset of RSL since CSP is a suitable vehicle for expressing suitably abstract synchronisation and communication between behaviours. (In Sect. 6.4.2 on Page 130 we bring, as an informative aside, The Petri Net Story.)

The mereology of domain parts induces channel declarations. CSP channels are loss-free. That is: two CSP processes, of which one offers and the other offers to accept a message do so synchronously and without forgetting that message. To model actual, so-called “real-life” communication via queues or allowing “channels” to forget, then you must model that explicitly in CSP. We refer to [238, 336, 340].

6.4.1 The CSP Story

CSP is a wonderful tool, i.e., a language with which to study and describe communicating sequential processes. It is the invention of Charles Anthony Richard Hoare. Major publications on CSP are [237, 239, 238, 336, 340].

6.4.1.1 Informal Presentation

CSP processes (models of domain behaviours) \( P_i, P_j, \ldots, P_k \) can proceed in parallel:

\[ P_j \parallel P_j \parallel \ldots \parallel P_k \]

Behaviours sometimes synchronise and usually communicate. Synchronisation and communication is abstracted as the sending \((ch!m)\) and receipt \((ch?m)\) of messages, \( m:M \), over channels, \( ch \).

\[
\begin{array}{c}
\text{type M} \\
\text{channel ch:M}
\end{array}
\]

Communication between (unique identifier) indexed behaviours have their channels modelled as similarly indexed channels:

\[
\begin{array}{c}
\text{out: } \text{ch[idx]!m} \\
\text{in: } \text{ch[idx]?} \\
\text{channel } \{ \text{ch[idx]:M|idx:IDE} \}
\end{array}
\]

where IDE typically is some type expression over unique identifier types.

The expression

\[ P_j \parallel P_j \parallel \ldots \parallel P_k \]

can be understood as a choice: either \( P_j \), or \( P_j \), or \( \ldots \) or \( P_k \) as non-deterministically internally chosen with no stipulation as to why.

The expression
can be understood as a choice: either $\mathbf{P}рес$, or $\mathbf{P}рес$, or ... or $\mathbf{P}рес$ as deterministically externally chosen on the basis that the one chosen offers to participate in either an input, $\mathbf{ch} ?$, or an output, $\mathbf{ch} ! \text{msg}$, event. If more than one $\mathbf{P}рес$ offers a communication then one is arbitrarily chosen. If no $\mathbf{P}рес$ offers a communication the behaviour halts till some $\mathbf{P}resa$ offers a communication.

### 6.4.1.2 A Syntax for CSP

We present the syntax for the CSP used in RSL.

$$
\begin{align*}
\mathbf{P} & ::= \text{stop} \\
& \mid \text{skip} \\
& \mid \mathbf{P} \parallel \mathbf{P} \quad \text{parallel composition (interleave)} \\
& \mid \mathbf{P} \parallel \mathbf{P} \quad \text{internal non-deterministic choice} \\
& \mid \mathbf{P} \parallel \mathbf{P} \quad \text{external non-deterministic choice} \\
& \mid \mathbf{P} ; \mathbf{P} \quad \text{sequential composition} \\
& \mid \text{if } \mathbf{B} \text{ then } \mathbf{P} \text{ else } \mathbf{P} \text{ end} \quad \text{Boolean conditional} \\
& \mid \text{let } v = \mathbf{ch} ? \text{ in } ... \text{ end} \quad \text{input value } v \text{ on channel } \mathbf{ch} \\
& \mid \mathbf{ch} ! e ; \mathbf{P} \quad \text{output value of expression } e \text{ on channel } \mathbf{ch}
\end{align*}
$$

### 6.4.1.3 Disciplined Uses of CSP

In connection with domain modelling, which uses of CSP appear to be meaningful? To understand our answer let us consider the following. As suggested in Chapters 3–4 the domain of endurants consists of a number of parts, some atomic, some compounded, that is, consisting of a part (a “root”) and a number of proper sub-parts (its “siblings”). With Chapter 5 we shall consider each and every part to also represent a behaviour, that is, with sub-parts representing behaviours not “embedded” in “root” part behaviours, but, in a “first approximation” only bound to their roots by mutual mereologies.

This is a modelling decision. We could have chosen a more elaborate one; one that, from the days of Algol 60 [257] was in line with the so-called ‘block structure’ concept. But have chosen not to!

---

**Example 78** We refer to Example 39 on Page 69. A bus company is like a “root” for its fleet of “sibling” buses. But a bus company may cease to exist without the buses therefore necessarily also ceasing to exist. They may continue to operate, probably illegally, without, possibly, a valid bus driving certificate. Or they may be passed on to either private owners or to other bus companies. We use this example as a reason for not endowing a “block structure” concept on behaviours.

So there we are. With a collection of part and sub-part behaviours that need communicate “across” and “within” compounds. To do so they avail themselves of channels, $\mathbf{ch}[i,j]$, output, $\mathbf{ch}[i,j] ! e$ and input, $\mathbf{ch}[i,j] ?$. The general situation is then that a number of behaviours, $\mathbf{P}рес$ and $\mathbf{Q}resa$, wishes to synchronise and communicate. The general, disciplined form for doing so can be schematically expressed as follows:

$$
\begin{align*}
\mathbf{P}(i,UIS,...)(...) & \equiv \\
& \ldots \\
& \bowtie \{ \mathbf{ch}[i,j] ! e ; ... | j \in UIS \} ; \\
& \ldots ; \\
& \mathbf{P}(i,UIS,...)(...) \\
\mathbf{Q}(j,UIS,...)(...) & \equiv \\
& \ldots \\
& \bowtie \{ \text{let } v = \mathbf{ch}[i,j] ? \text{ in } ... \text{ end } | i \in UIS \} ; \\
& \ldots \\
& \mathbf{Q}(j,UIS,...)(...) 
\end{align*}
$$
The \( \boxdot \) operator is either \( \lceil \), or \( \rfloor \), or \( \lceil \). We shall abstain from further ‘advice’ on the use of CSP but refer to either [39, Software Engineering 2, Chapter 21] (Concurrent Specification Programming) or standard CSP textbooks [237, 239, 238, 336, 340]. We shall take up this line of inquiry in Sect. 6.7.8 on Page 146 – A Suggested Behaviour Definition 2 on Page 146.

6.4.2 The Petri Net Story

Petri nets\(^2\) are a wonderful concept first invented by Carl Adam Petri [314]. It is intended to model a class of discrete event dynamic systems. A Petri net is a directed bipartite graph, in which some nodes (traditionally represented by bars) represent transitions (i.e. events) that may occur, and other nodes represent places (i.e. conditions, traditionally represented by circles). The directed arcs describe which places are pre- and/or post-conditions for which transitions (signified by arrows). We shall basically recommend the Petri net books by Wolfgang Reisig, some of which are [328, 329, 332, 333] – notably [333].

6.4.2.1 Informal Presentation

Figure 6.1 shows a simplest form of Petri Net. Let us focus on the left net. The labeled circles designate places. The labeled thick, black bar designate a transition. The arrows, \( \rightarrow \), designate (flow) arcs and are labeled with a numeral, designating a natural number larger than 0. Inside the places we show 2, 2, respectively 0 tokens. Their constellation is also called a marking. In general, any composition of places, transitions, markings and labels such that arcs emanating from a place are incident upon transitions and such that arcs emanation from transitions are incident upon places, form a syntactically meaningful Petri net, also called a Place-Transition Net, PTN.

![Fig. 6.1. Two Petri Nets: Before and after Firing](image)

Let us start by focusing on the left Petri net. The meaning of the number of tokens in places, the transition input arc labels, and the transition output arc labels are as follows: If the respective transition input arc labels can be satisfied by the respective number of tokens in source places then a firing can take place. After a firing the Petri net has a new constellation.

The following (two-and-a-quarter pages)\(^3\) was written by Christian Krogh Madsen (around 2004)\(^4\)

6.4.2.2 An Example – Christian Krogh Madsen

Critical Resource Sharing

\(^2\) The term Petri net stands for the ‘language’ of Petri nets. A Petri net is an instance of the language of Petri nets.

\(^3\) – an An Example and An RSL Model of Petri nets.

\(^4\) Christian Krogh Madsen devised and wrote Chapters 12–14: Petri Nets, Message and Live Sequence Charts, and Statecharts in [40, Pages 315–508].
6.4 Modelling Concurrent Behaviours

![Petri Nets Diagram](image)

**Fig. 6.2. Critical resource sharing**

**Example 79** Figure 6.2 shows an example PTN modelling four processes that access a common critical resource. One process writes to the resource, while the other three processes read from the resource. To ensure data integrity, mutual exclusion must be enforced between the writing process and the reading processes. The protocol for mutual exclusion requires a reading process to claim a key before it may read, while the writing process is required to claim three keys before it may write. A process that cannot get the required number of keys must wait until more keys become available. The place Keys holds a token for each key that is unused. When a process finishes reading or writing it returns the claimed keys to the place Keys and proceeds to do some processing that does not access the critical resource.

### 6.4.2.3 An RSL Model of Petri nets – Christian Krogh Madsen

#### 6.4.2.3.1 Syntax of Petri nets

We first formalise a syntax and then a static semantics for Petri nets, as PTN (for place-transition net), with finite capacity places.

- A place transition net consists of a set of places with associated capacities, a set of transitions, a preset, a postset and a marking.
- Only well-formed PTNs will be considered.
- Places and transitions are further unspecified entities.
- Presets are a mapping from transitions to sets of pairs of places and weights.
- Postsets are a mapping from transitions to sets of pairs of places and weights.
- A marking is a mapping of places to marks.
- A mark is a non-negative integer.

**type**

\[
PTN = \{ | ptn:PTN \cdot wf_{PTN}(ptn) | \}
\]

\[
PTN' = (Place \rightarrow \mathbb{N}) \times Trans-set \times Preset \times Postset \times Marking
\]

**Place**

**Trans**

**Preset** = Trans \rightarrow (Place \times \mathbb{N})-set

**Postset** = Trans \rightarrow (Place \times \mathbb{N})-set

**Marking** = Place \rightarrow \mathbb{N}

#### 6.4.2.3.2 A Static Semantics

- A PTN is well-formed if:
1-2 every transition in the set of transitions is included in the domain of the maps of presets and
postsets, and
3 every place is in the pre- or postset of some transition, and
4 every transition has a non-empty preset or postset, and
5 no transition can have a preset or postset that includes the same place more than once with different
weights, and
6 the marking covers all places, and
7 for every place the number of tokens assigned to it in the marking must be at most equal to the
capacity of the place.

value

\[ \text{wf}_{PTN} : PTN' \rightarrow \text{Bool} \]
\[ \text{wf}_{PTN}(ps, ts, pres, posts, mark) \equiv \]
\[ [1] \text{dom pres} = ts \land \]
\[ [2] \text{dom posts} = ts \land \]
\[ [3] \{ p \mid p: \text{Place} \cdot \]
\[ \exists \text{pns: (Place} \times \text{Nat})\text{-set}, n: \text{Nat} \cdot \]
\[ (p,n) \in \text{pns} \land \text{pns} \in \text{rng pres} \cup \text{rng posts} \} = \text{dom ps} \land \]
\[ [4] (\forall t: \text{Trans} \cdot t \in ts \Rightarrow \text{pres}(t) \cup \text{posts}(t) \neq \{ \}) \land \]
\[ [5] (\forall t: \text{Trans} \cdot \]
\[ \sim (\exists n_1, n_2 : \text{Nat}, p : \text{Place} \cdot \]
\[ n_1 \neq n_2 \land p \in \text{dom ps} \land \]
\[ (\{ (p,n_1), (p,n_2) \} \subseteq \text{pres}(t) \lor \]
\[ (\{ (p,n_1), (p,n_2) \} \subseteq \text{posts}(t))) \land \]
\[ [6] \text{dom mark} = \text{dom ps} \land \]
\[ [7] (\forall p: \text{Place} \cdot p \in \text{dom ps} \Rightarrow \text{mark}(p) \leq ps(p)) \]

6.4.2.3.3 A Dynamic Semantics

We formalise the dynamic aspects of PTN, namely what it means for a transition to be activated and for a
transition to occur.

• A transition is activated:
  \( \sim \) if for every place in its preset there are at least as many tokens as the weight of the corresponding
  arrow, and
  \( \sim \) if for every place in its postset the number of tokens at that place added to the weight of the corre-
  sponding arrow is at most equal to the capacity of the place.
• The occurrence of an activated transition produces a new marking
  \( \sim \) in which the number of tokens at each of the places in the preset is reduced by the weight of the corre-
  sponding arrow, and
  \( \sim \) in which the number of tokens at each of the places in the postset is increased by the weight of the corre-
  sponding arrow.

value

\[ \text{activated} : \text{Trans} \times \text{PTN} \rightarrow \text{Bool} \]
\[ \text{activated}(t,ptn) \equiv \]
\[ \text{let (ps,ts, pres,posts,mark) = ptn in} \]
\[ (\forall p: \text{Place}, n: \text{Nat} \cdot (p,n) \in \text{pres}(t) \Rightarrow \text{mark}(p) \geq n) \land \]
\[ (\forall p: \text{Place}, n: \text{Nat} \cdot (p,n) \in \text{posts}(t) \Rightarrow \text{mark}(p) + n \leq ps(p)) \]
\[ \text{end} \]
\[ \text{pre let (ps,ts, pres,posts,mark) = ptn in t \in ts end} \]
6.5 Channels and Communication

6.5.1 From Mereologies to Channel Declarations

The fact that a part \( p \) of sort \( P \) with unique identifier \( p_i \), has a mereology, for example the set of unique identifiers \( \{q_a, q_b, ..., q_d\} \) identifying parts \( \{q_a, q_b, ..., q_d\} \) of sort \( Q \), may mean that parts \( p \) and \( \{q_a, q_b, ..., q_d\} \) may wish to exchange – for example, attribute – values, one way (from \( p \) to the \( q_s \)) or the other (vice versa) or in both directions.

Figure 6.3 shows two dotted rectangle box diagrams.

**Fig. 6.3.** Respective Part and Behaviour/Channel Constellations. \( u:p \) unique id. \( p \); \( m:p \) mereology \( p \)

The left fragment of Fig. 6.3 intends to show a 1:1 Constellation of a single \( p:P \) box and a single \( q:Q \) part, respectively, indicating, within these parts, their unique identifiers and mereologies. The right fragment of the figure intends to show a 1:n Constellation of a single \( p:P \) box and a set of \( q:Q \) parts,
now with arrowed lines connecting the \( p \) part with the \( q \) parts. These lines are intended to show channels. We show them with two way arrows. We could instead have chosen one way arrows, in one or the other direction. The directions are intended to show a direction of value transfer. We have given the same channel names to all examples, \( \text{ch}_{PQ} \). We have ascribed channel message types \( MPQ \) to all channels.\(^5\) Figure 6.4 shows an arrangement similar to that of Fig. 6.3 on the preceding page, but for an \( m:n \) Constellation.

**Fig. 6.4.** Multiple Part and Channel Arrangements: \( u:p \) unique id; \( m:p \) mereology \( p \)

The channel declarations corresponding to Figs. 6.3 and 6.4 are:

\[
\text{channel}
\]

\[
\begin{align*}
\text{channel} & \{ \text{ch}_{PQ}[i,j]:MPQ \\
\text{channel} & \{ \text{ch}_{PQ}[i,x]:MPQ \ | \ x: \{j,k,\ldots,l\} \} \\
\text{channel} & \{ \text{ch}_{PQ}[p,q]:MPQ \ | \ p: \{x,y,\ldots,z\}, q: \{j,k,\ldots,l\} \} \\
\end{align*}
\]

Since there is only one index \( i \) and \( j \) for channel [1], its declaration can be reduced. Similarly there is only one \( i \) for declaration [2]:

\[
\text{channel}
\]

\[
\begin{align*}
\text{channel} & \{ \text{ch}_{PQ}:MPQ \\
\text{channel} & \{ \text{ch}_{PQ}[x]:MPQ \ | \ x: \{j,k,\ldots,l\} \} \\
\end{align*}
\]

\(^{233}\) The following description identities holds:

\[
\begin{align*}
233 \ { \text{ch}_{PQ}[x]:MPQ \ | \ x: \{j,k,\ldots,l\} } \equiv \text{ch}_{PQ}[j],\text{ch}_{PQ}[k],\ldots,\text{ch}_{PQ}[l], \\
233 \ { \text{ch}_{PQ}[p,q]:MPQ \ | \ p: \{x,y,\ldots,z\}, q: \{j,k,\ldots,l\} } \equiv \\
233 \ { \text{ch}_{PQ}[x,j],\text{ch}_{PQ}[x,k],\ldots,\text{ch}_{PQ}[x,l], \\
233 \ { \text{ch}_{PQ}[y,j],\text{ch}_{PQ}[y,k],\ldots,\text{ch}_{PQ}[y,l], \\
233 \ { \ldots, \\
233 \ { \text{ch}_{PQ}[z,j],\text{ch}_{PQ}[z,k],\ldots,\text{ch}_{PQ}[z,l] }
\end{align*}
\]

We can sketch a diagram similar to Figs. 6.3 on the previous page and 6.4 for the case of composite parts.

**6.5.2 Channel Declarations**

We can simplify the general treatment of channel declarations. Basically all we can say, for any domain, is that any two distinct part behaviours may need to communicate. Therefore we declare a vector of channels indexed by sets of two distinct part identifiers.

\[
\text{channel} \{ \text{ch}[\{ij,ik\}] \ | \ ij,ik\notin\text{all}\text{uniq}\text{id}\text{s}() \land ij\neq ik \} \ M
\]

\(^{5}\) Of course, these names and types would have to be distinct for any one domain description.
Initially we shall leave the type of messages over channels further undefined. As we, laboriously, work through the definition of behaviours, Sect. 6.7, we shall be able to make \( M \) precise. All \( \texttt{uniq ids} \) was defined in Sect. 4.2.4 on Page 87.

In preparation for the next example we show Figure 6.5. In that example we shall however refine the channel declaration indices to two element sets of unique identifiers from specific part identifier types.

**Example 80**  We shall argue for hub-to-link channels based on the mereologies of those parts. Hub parts may be topologically connected to any number, 0 or more, link parts. Only instantiated road nets knows which. Hence there must be channels between any hub behaviour and any link behaviour. Vice versa: link parts will be connected to exactly two hub parts. Hence there must be channels from any link behaviour to two hub behaviours. See the figure above.

**Channel Message Types:**

We ascribe types to the messages offered on channels.

234 Hubs and links communicate, both ways, with one another, over channels, \( \texttt{hl\_ch} \), whose indexes are determined by their mereologies.

235 Hubs send one kind of messages, links another.

236 Bus companies offer timed bus time tables to buses, one way.

237 Buses and automobiles offer their current, timed positions to the road element, hub or link they are on, one way.

**Channel Declarations:**

This justifies the channel declaration which is calculated to be:

238 This justifies the channel declaration which is calculated to be:

\[
\text{channel} \\
\{ \text{hl\_ch}[\texttt{\{\texttt{h\_ui},\texttt{\texttt{l\_ui}\}\}}]:\texttt{H\_L\_Msg} \\
\mid \text{h\_ui}\in\texttt{h\_uis} \land \text{h\_ui}\in\texttt{lh\_ums}(\text{h\_ui}) \} \\
\}
\]

\[
\{ \text{hl\_ch}[\texttt{\{\texttt{h\_ui},\texttt{\texttt{l\_ui}\}\}}]:\texttt{L\_H\_Msg} \\
\mid \text{h\_ui}\in\texttt{h\_uis} \land \text{h\_ui}\in\texttt{lh\_ums}(\text{h\_ui}) \} \\
\}
\]

\[
\{ \text{v\_r\_ch}[\texttt{\{\texttt{h\_ui},\texttt{\texttt{l\_ui}\}\}}]:\texttt{V\_R\_Msg} \\
\mid \text{h\_ui}\in\texttt{h\_uis} \land \text{h\_ui}\in\texttt{lh\_ums}(\text{h\_ui}) \} \\
\}
\]

\[
\{ \text{bc\_b\_ch}[\texttt{\{\texttt{h\_ui},\texttt{\texttt{l\_ui}\}\}}]:\texttt{BC\_B\_Msg} \\
\mid \text{h\_ui}\in\texttt{h\_uis} \land \text{h\_ui}\in\texttt{lh\_ums}(\text{h\_ui}) \} \\
\}
\]
We shall argue for bus company-to-bus channels based on the mereologies of those parts. Bus companies need communicate to all its buses, but not the buses of other bus companies. Buses of a bus company need communicate to their bus company, but not to other bus companies.

This justifies the channel declaration which is calculated to be:

\[
\begin{align*}
\text{channel} & \quad \{ \text{bc}_{\text{ui}, \text{b}} | \text{bc}_{\text{ui}, \text{b}} \in \text{bc}_{\text{ui}s} \} : \text{BC}_{\text{B}}\text{Msg} \\
\text{bc}_{\text{ui}} & \in \text{bc}_{\text{ui}s} \land \text{b}_{\text{ui}} \in \text{b}_{\text{ui}s}
\end{align*}
\]

We shall argue for vehicle-to-road element channels based on the mereologies of those parts. Buses and automobiles need communicate to all hubs and all links.

This justifies the channel declaration which is calculated to be:

\[
\begin{align*}
\text{channel} & \quad \{ \text{v}_{\text{ui}, \text{r}} | \text{v}_{\text{ui}, \text{r}} \in \text{v}_{\text{ui}s} \} : \text{V}_{\text{R}}\text{Msg} \\
\text{v}_{\text{ui}} & \in \text{v}_{\text{ui}s} \land \text{r}_{\text{ui}} \in \text{r}_{\text{ui}s}
\end{align*}
\]

The channel calculations are described on Pages 139–140.

### 6.6 Signatures – In General

We shall treat perdurants as function invocations. In our cursory overview of perdurants we shall now focus on one perdurant quality: function signatures.

**Definition:** 68 **Function Signature:** By a function signature we shall understand a function name and a function type expression.

**Definition:** 69 **Function Type Expression:** By a function type expression we shall understand a pair of type expressions separated by a function type constructor either \(\rightarrow\) (for total function) or \(\sim\rightarrow\) (for partial function).

The type expressions are part sort or type, or material sort or type, or attribute type names, but may, occasionally be expressions over respective type names involving \(-set, \times, ^*, \, \&\) and \(\mid\) type constructors.

### 6.6.1 Action Signatures and Definitions

Actors usually provide their initiated actions with arguments, say of type \(\text{VAL}\). Hence the schematic function (action) signature and schematic definition:

\[
\begin{align*}
\text{action}: & \quad \text{VAL} \rightarrow \Sigma \xrightarrow{\sigma'} \Sigma \\
\text{action}(v)(\sigma) & \text{ as } \sigma' \\
\text{pre}: & \quad \mathcal{P}(v,\sigma) \\
\text{post}: & \quad \mathcal{Q}(v,\sigma,\sigma')
\end{align*}
\]

expresses that a selection of the domain state, as designated by the \(\Sigma\) type expression, is acted upon and possibly changed. The partial function type operator \(\sim\rightarrow\) shall indicate that \(\text{action}(v)(\sigma)\) may not be defined for the argument, i.e., initial state \(\sigma\) and/or the argument \(v:\text{VAL}\), hence the precondition \(\mathcal{P}(v,\sigma)\). The post condition \(\mathcal{Q}(v,\sigma,\sigma')\) characterises the “after” state, \(\sigma':\Sigma\), with respect to the “before” state, \(\sigma:\Sigma\), and possible arguments \((v:\text{VAL})\). Which could be the argument values, \(v:\text{VAL}\), of actions? Well, there can basically be only the following kinds of argument values: parts, components and materials, respectively unique part identifiers, mereologies and attribute values.
Perdurant (action) analysis thus proceeds as follows: identifying relevant actions, assigning names to these, delineating the "smallest" relevant state\(^6\), ascribing signatures to action functions, and determining action pre-conditions and action post-conditions. Of these, ascribing signatures is the most crucial: In the process of determining the action signature one oftentimes discovers that part or component or material attributes have been left ("so far") "undiscovered".

6.6.2 Event Signatures and Definitions:

Events are usually characterised by the absence of known actors and the absence of explicit "external" arguments. Hence the schematic function (event) signature:

\[
\text{value} \\
\text{event: } \Sigma \times \Sigma \Rightarrow \text{Bool} \\
\text{event}(\sigma, \sigma') \text{ as if} \\
\text{pre: } P(\sigma) \\
\text{post: } \text{tf} = Q(\sigma, \sigma')
\]

The event signature expresses that a selection of the domain as designated by the \(\Sigma\) type expression is "acted" upon, by unknown actors, and possibly changed. The partial function type operator \(\Rightarrow\) shall indicate that \(\text{event}(\sigma, \sigma')\) may not be defined for some states \(\sigma\). The resulting state may, or may not, satisfy axioms and well-formedness conditions over \(\Sigma\) – as expressed by the post condition \(Q(\sigma, \sigma')\).

Events may thus cause well-formedness of states to fail. Subsequent actions, once actors discover such "disturbing events", are therefore expected to remedy that situation, that is, to restore well-formedness. We shall not illustrate this point.

6.6.3 Behaviour Signatures

We shall only cover behaviour signatures when expressed in RSL/CSP [176]. The behaviour functions are now called processes. That a behaviour function is a never-ending function, i.e., a process, is "revealed" by the "trailing" Unit:

\[
\text{behaviour: } \ldots \Rightarrow \text{Unit}
\]

That a process takes no argument is "revealed" by a "leading" Unit:

\[
\text{behaviour: } \text{Unit} \Rightarrow \ldots
\]

That a process accepts channel, viz.: \(\text{ch}\), inputs, is "revealed" as follows:

\[
\text{behaviour: } \ldots \Rightarrow \text{in ch} \ldots
\]

That a process offers channel, viz.: \(\text{ch}\), outputs is "revealed" as follows:

\[
\text{behaviour: } \ldots \Rightarrow \text{out ch} \ldots
\]

That a process accepts other arguments is "revealed" as follows:

\[
\text{behaviour: } \text{ARG} \Rightarrow \ldots
\]

where \(\text{ARG}\) can be any type expression:

\[
T, T\Rightarrow T, T\Rightarrow T, T, T\Rightarrow T, T, \ldots
\]

where \(T\) is any type expression.

\(^{6}\) By "smallest" we mean: containing the fewest number of parts. Experience shows that the domain analyser cum describer should strive for identifying the smallest state.
6.6.4 Attribute Access, An Interpretation

We shall only be concerned with part attributes. And we shall here consider them in the context of part behaviours. Part behaviour definitions embody part attributes.

- **Static attributes** designate constants. As such they can be “compiled” into behaviour definitions. We choose, thus, to bring static attribute values as explicit behaviour arguments.
- **Monitorable-only attributes** designate time-varying values whose values we choose to access in the following manner:
  \[ \text{attr}_A(p) \]
  where \( p \) is a part in the global state, cf. Sect. 3.15 on Page 61.
- **Biddable attributes** designate time-varying values whose values we choose to access, respectively biddably \( \text{update} \) in the following manner:
  \[ \text{attr}_A(p) \]
  \[ \text{update}(\text{attr}_A,A,p) \]
  where \( p \) is a part in the global state. We shall informally explain the \( \text{update} \) functional below.
- **Programmable attribute** values are calculated by their behaviours. We list them as behaviour arguments. The behaviour definitions may then specify new values. These are provided in the position of the programmable attribute arguments in tail recursive invocations of these behaviours.

6.6.4.1 The \( \text{update} \) Functional

The generic \( \text{update} \) function is explained very informally:

\[ 1 \] \( \text{update} : (P \rightarrow A) \times A \times P \rightarrow P \)
\[ 2 \] \( \text{update}(\text{attr}_A,a,p) \equiv p' \)
\[ 3 \] \( \text{pre} \ A \in \text{analyse_attributes}(p) \land \text{parts}[\text{uid}_P(p)] \text{ is declared} \land \ldots \)
\[ 4 \] \( \text{post} \ \text{attr}_A(p') \approx a \land \ldots \)

[1] The first argument is the observe attribute function, the second argument is the attribute value, the third argument is the part, \( p \), being \text{updated}.
[2] The result of applying the \( \text{update} \) function is a part, \( p' \).
[3] The pre-condition is that the attribute type, \( A \), is amongst the attributes of the part, that part \( p \) is in the global state, i.e., has been declared as a variable, and more!
[4] The post-condition is that the \text{updated} attribute of \( p' \) approximates the argument attribute value, and “much, much more”.

The “much, much more” refers to the following: the unique identifier of \( p' \) is that of \( p \); the mereology of \( p' \) is that of \( p \); all other attribute values of \( p' \) are those of \( p \); and no other part has changed values.

The above amounts to a “storage model”, i.e., a model of domain state variables akin to the storage models put forward first in [26, Bekić and Walk, 1971], see also [22, Bekić, Bjørner, Henhapl, Jones and Lucas, 1974], then in [40, Sect. 8.7.1, Bjørner, 2006].

In the context of domain models we shall (later) introduced an array, \( \text{parts} \), of variables global to an entire domain description. For each physical part, \( p \), with unique identifier, \( \pi \), there will be a corresponding array element: \( \text{parts}[\pi] \). To obtain a monitorable attribute \( A \) value for part \( p \)
- is thus expressed as \( \text{attr}_A(\text{parts}[\pi]) \).

To update a monitorable attribute \( A \) to value \( a:A \) for part \( p \)
- is correspondingly expressed as \( \text{update}(\text{attr}_A,a,\text{parts}[\pi]) \).
6.6.4.2 Calculating In/Output Channel Signatures:

The function \texttt{calc\_chn\_refs} apply to parts and yield RSL-Text.

- From \(p\) we calculate its unique identifier value and its mereology value.
- If the mereology is not void then a (Currying\(^7\)) right pointing arrow, \(\rightarrow\), is inserted.\(^8\)
- If there is an input mereology then the keyword \texttt{in} is inserted in front of the input mereology;
- similarly for the \texttt{input}/\texttt{output} mereology;
- and for the \texttt{output} mereology.

\[
\text{value} \quad \texttt{calc\_chn\_refs}(p) = \begin{cases} 
\text{uid}_P(p), & \text{ics, iocs, ocs} \\
\text{obs\_mereo}(p) & \text{ics} \cup \text{iocs} \cup \text{ocs} \cup \text{atrvs} \neq \emptyset \\
\rightarrow & \text{ics} \neq \emptyset \\
\texttt{in} & \text{ics} \neq \emptyset \\
\texttt{in, out} & \text{iocs} \neq \emptyset \\
\texttt{out} & \text{ocs} \neq \emptyset
\end{cases}
\]

The \texttt{calc\_chn\_dcls} function apply to a pair, \((pui, quis)\) of a unique part identifier and a set of unique part identifiers and yield RSL-Text.

- If \(quis\) is empty no text is generated. Otherwise an array channel declaration is generated.

\[
\text{value} \quad \texttt{calc\_chn\_dcls}(pui, quis) = \begin{cases} 
\text{\((pui, qui)\) ch[pui, qui]} & \text{quis} \\
\text{\((pui, qui)\) ch[pui, qui]} : \eta(pui, qui)M & \text{quis} \\
\end{cases}
\]

The overloaded distributed-fix operator \(\eta\)\(^9\) is here applied to a pair of unique identifiers. Very informally:

\[
\text{value} \quad \texttt{calc\_chn\_refs} : P \rightarrow \text{RSL-Text}
\]

\[
\text{value} \quad \texttt{calc\_chn\_dcls} : P \times Q - \text{set} \rightarrow \text{RSL-Text}
\]

The \(\eta\) operator applies to a type and yields the name of the type.

\[
\text{value} \quad \texttt{calc\_chn\_refs} : P \rightarrow \text{RSL-Text}
\]

\[
\text{value} \quad \texttt{calc\_chn\_dcls} : P \times Q - \text{set} \rightarrow \text{RSL-Text}
\]

\[
\text{value} \quad \texttt{calc\_chn\_refs} : (U, UI) \rightarrow \text{RSL-Text}
\]

\[
\text{value} \quad \texttt{calc\_chn\_dcls} : (X, UI) \times (Y, UI) \rightarrow \text{RSL-Text}
\]

Repeating these channel calculations over distinct parts \(p_1, p_2, \ldots, p_n\) of the same part type \(P\) will yield “similar” behaviour signature channel references:

\(^7\) https://en.wikipedia.org/wiki/Currying
\(^8\) We refer to the three parts of the mereology value as the input, the input/output and the output mereology (values).
\(^9\) The \(\eta\) operator applies to a type and yields the name of the type.
6.7 Behaviour Signatures and Definitions

In this section we shall finally show the schemas whereby discrete endurants are transcendentally “morphed” into behaviours.

6.7.1 General on Behaviour Schemas

The general translation schema can be expressed as follows:

\[
\text{Translation Schema 2}
\]

\[
\text{value is\_endurant}(e) \\
\]

\[
\begin{align*}
\text{BEHAVIOUR} & \text{Endurant}: E \rightarrow \text{RSL-Text} \\
\text{BEHAVIOUR} & \text{Endurant}(e) \equiv \\
\text{is\_physical\_part}(e) & \rightarrow \\
3 \text{ Pg. 141} & \text{is\_atomic}(e) \rightarrow \text{BEHAVIOUR}\text{Atomic}(e), \\
4 \text{ Pg. 142} & \text{is\_composite}(e) \rightarrow \text{BEHAVIOUR}\text{Composite}(e), \\
5 \text{ Pg. 142} & \text{is\_conjoin}(e) \rightarrow \text{BEHAVIOUR}\text{Conjoin}(e), \\
\text{is\_living\_species}(e) & \rightarrow \text{skip} \\
\end{align*}
\]

We have chosen to not “morph” materials into behaviours – as expressed by the last clause above.

6.7.1.1 The General Behaviour Signature

We associate with each part, \( p: P \), a behaviour name \( \mathcal{M}_p \). That is, every part \( p \) of sort \( P \) is associated with the same behaviour name \( \mathcal{M}_p \) each individual such behaviour being distinguished by the initial unique identifier constant argument.

Behaviours thus have as their first argument their unique part identifier: \( \text{uid}_P(p) \). Behaviours evolves around a state, or, rather, a set of values: its possibly changing mereology, \( \text{mt:MT} \) and the attributes of the part.\(^{10}\) A behaviour signature is therefore:

\[
\begin{align*}
\mathcal{M}_p: \text{ui:UI} \times \text{me:MT} \times \text{stat\_attr}\_types(p) \\
\text{prgr\_attr}\_types(p) \rightarrow \text{calc\_io\_chn}\_refs(p) \rightarrow \text{Unit}
\end{align*}
\]

\(^{10}\) We presently leave out consideration of possible components and materials of the part.
where (i) $ui:Ul$ is the unique identifier value and type of part $p$; (ii) $me:MT$ is the value and type mereology of part $p$, $me = meroo_P(p)$; (iii) $stat\_attr\_types(p)$: static attribute types of part $p: P$; (iv) $prgr\_attr\_types(p)$: controllable attribute types of part $p: P$; (v) $calc\_io\_chn\_refs(p)$ calculates references to the input, the input/output and the output channels serving the attributes shared between part $p$ and the parts designated in its mereology $me$.

### 6.7.2 Preamble Definitions

We have, in Chapter 4 and in this chapter, defined a number of analysis predicates, analysis functions, and perdurant calculators. These will be used in the preamble of all the part Behaviour Schemas of this section. We summarise some relevant functions and perdurant calculators.

- $calc\_all\_chn\_dcls$, Item 243a, 139
- $calc\_chn\_refs$, Item 242a, 139
- $calc\_io\_chn\_refs$, Item 241, 139
- $declaring\_all\_monitorable\_variables$, Item 229, 126
- $moni\_attr\_types$, Item 183, 101
- $moni\_attr\_vals$, Item 186, 101
- $prgr\_attr\_types$, Item 184, 101
- $prgr\_attr\_vals$, Item 188, 101
- $stat\_attr\_types$, Item 182, 101
- $stat\_attr\_vals$, Item 186, 101
- Translate Endurant, 140

Each Behaviour Schema requires more-or-less all of the below:

- UI, unique identifier type;
- MT, mereology type;
- ST, static attribute types;
- PT, programmable attribute types;
- IOR, input/output channel references.

### 6.7.3 A Behaviour Signature Calculator

For each endurant to be Behaviour, we need collect the elements, values and types that are relevant to that endurant’s behaviour signature.

$$
\text{collect\_signature}: E \rightarrow \langle UI, MT, ST, PT, IOR \rangle
$$
$$
\text{collect\_signature}(e) \equiv \\
(type\_of(uid\_E(e)), type\_of(mereo\_E(e)), \\
stat\_attr\_types(e), prgr\_attr\_types(e), \\
calc\_io\_chn\_refs(e))
$$

So we assume this clause to be part of each $e:E$ schema, ... below:

```plaintext
value
let ($\langle UI, MT, ST, PT, IOR \rangle$) = collect\_signature(e) in ... end
```

The Behaviour schemas that now follow make use of analyse part materials part and analyse part materials endurant analysis function prompts defined in Sect. 3.17 on Page 63.

### 6.7.4 Atomic Schema

Let $p:P$ be an atomic part. It “translates” into behaviour $\mathcal{M}_p$:

```
is\_atomic(e)
```
6.7.5 Composite Schema

Let $P$ be a composite sort defined in terms of endurant sub sorts $E_1, E_2, \ldots, E_n$. Here we only need be concerned with the translation of $p:P$, translation of "siblings" follows from the sub-sort endurants $\theta_1, \theta_2, \ldots, \theta_n$ which have been set aside. The behaviour description translated from $p:P$, is thus a behaviour description of the "root", $M_p$, relying on and handling the unique identifier, mereology and attributes of part $p$.

Translation Schema 4

\[
\text{BEHAVIOUR}_{\text{Composite}}(\theta) \equiv \text{value}
\]

\[
.M_p: \text{UI} \times \text{MT} \times \text{ST} \rightarrow \text{PT} \rightarrow \text{IOR} \ \text{Unit}
\]

\[
.M_p(\text{ui}, \text{me}, \text{sv})(\text{pv}) \equiv B_p(\text{ui}, \text{me}, \text{sv})(\text{pv})
\]

6.7.6 Structure Schema

Translation Schema 5

\[
\text{BEHAVIOUR}_{\text{Structure}}: E \rightarrow \text{RSL-Text}
\]

\[
\text{BEHAVIOUR}_{\text{Structure}}(\theta) \equiv \text{value}
\]

\[
.M_E: \text{UI} \times \text{MT} \times \text{ST} \rightarrow \text{PT} \rightarrow \text{IOR} \ \text{Unit}
\]

\[
.M_E(\text{ui}, \text{me}, \text{sv})(\text{pv}) \equiv B_E(\text{ui}, \text{me}, \text{sv})(\text{pv})
\]

6.7.7 Conjoin Schemas

Translation Schema 6

\[
\text{BEHAVIOUR}_{\text{Conjoin}}: E \rightarrow \text{RSL-Text}
\]

The signature identifiers UI, MT, ST and PT are taken from the collect_signature function. They are always understood syntactically when "occurring" in, e.g., signatures. Expression $B_p(\text{ui}, \text{me}, \text{sv})(\text{pv})$ stands for the behaviour definition body in which the names ui, me, sv, pv are chosen, freely by the domain describer and bound to the behaviour definition head, i.e., the left hand side of the $\equiv$. That expression, $B_p(\text{ui}, \text{me}, \text{sv})(\text{pv})$, may thus stand for quite a complex RSL/CSP clause. We elaborate on that in Sect. 6.7.8.

6.7.8 Modelling Choice 7 Composites: The above schema mandates that the conjoin behaviour, $M_E$, be defined. It does not say anything about the subsidiary elements of the composite. They are handled by the analyse and describe_perdurant process, Sect. 6.12 on Page 155. Why do we express the above? We do so because the schemas are just suggestions! The domain analyser & describer method mandates that all observed parts be described.
## 6.7 Behaviour Signatures and Definitions

### Behaviour Signatures

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Behaviour_conjoin(e)</code></td>
<td>Follows the signature of the part conjoin.</td>
</tr>
<tr>
<td><code>Behaviour_part_materials_conjoin(e)</code></td>
<td>Follows the signature of the part-materials conjoin.</td>
</tr>
<tr>
<td><code>Behaviour_material_parts_conjoin(e)</code></td>
<td>Follows the signature of the material-parts conjoin.</td>
</tr>
</tbody>
</table>

### 6.7.7.1 The Part-Materials Conjoin Schema

The **Part-Materials Conjoin Schema** reveals more of the “semantics” of conjoins. A part-materials conjoin gives rise to one behaviour: the conjoin behaviour, $\mathcal{M}_C$, with the additional programmable-like argument of the conjoin material. That is, in this monograph, we shall treat materials as “passive”, i.e., not having a behaviour that we define separately from that of $\mathcal{M}_C$.

**Translation Schema 7**

```plaintext
let (\_,(M1,..,Mm)) = analyse_part_materials_materials(e) in
value
Behaviour_part_materials_conjoin(e) ≡
  value
  \mathcal{M}_C: UI×MT×ST → PT×(M1×..×Mm) → IOR Unit
  \mathcal{M}_C(ui,me,sv)(pv,cm) ≡ B_P(ui,me,sv)(pv,cm)
```

### 6.7.7.2 The Material-Parts Conjoin Schema

The **Material-Parts Conjoin Schema** reveals more of the “semantics” of conjoins. A material-parts conjoin gives rise to one behaviour: the conjoin “root” behaviour, $\mathcal{M}_C$. The behaviour of the “sibling” part behaviours is defined separately – as is expressed by the `analyse` and `describe_perdurant_process` of Sect. 6.12 on Page 155. The former, $\mathcal{M}_C$, “keeps track” of the material compound, cm, relating the contained materials to the atomic “root” part. The “sibling” behaviours proceed at their own will.

**Translation Schema 8**

```plaintext
let (_,CM) = analyse_material_parts_material(e) in
value
Behaviour_material_parts_conjoin(e) ≡
  value
  \mathcal{M}_C: UI×MT×ST → PT×CM IOR Unit
  \mathcal{M}_C(ui,me,sv)(pv,cm) ≡ B_P(ui,me,sv)(pv,cm)
```

### Modelling Choice 8 Material-Parts:

The `analyse` and `describe_perdurant_process`, cf. Sect. 6.12 on Page 155 does prescribe the schema for some arbitrarily chosen part in ps, that is, mandates that all be described – and that is why we are mentioning it here.

### A Conjoin Canal Lock

**Example 81** Let $p$ be a conjoin canal lock with atom part $a$ and material $m$. Then $m$ is the water, a natural material, in the conjoin, housed in the fixed chamber of $p$, and $a$ is the lock mechanics: two gates
that can open and close, letting water in and out of the lock, a paddle, i.e., a valve by means of which water is filled or emptied, a winding gear, the mechanism which allows paddles to be lifted (opened) or lowered (closed), et cetera. The $M_c$ behaviour, i.e., the overall behaviour of the canal lock, when it so decides\footnote{We do not model the vessels that travels the canals and enter and leave locks.} inform the $M_A$ behaviour to operate its mechanics; it does so based on either sampling its container, $m$, water level, say by means of a dynamic attribute $\text{attr}_\text{level}(m)$, or receiving appropriate messages from the $M_A$ behaviour. The $M_A$ behaviour, i.e., the lock mechanics, in a sense, is oblivious to the water (and the vessels), and keeps itself occupied by monitoring and controlling its various mechanisms: the gates, paddles, winding gear, et cetera.

6.7.7.3 The Part-Parts Conjoin Schema

The Part-Parts Conjoin Schema reveals more of the “semantics” of conjoins. A part-parts conjoin gives rise to one behaviour: the conjoin’s “root” part behaviour, $M_C$. $M_C$ may be expected to “keep track” of the “sibling” parts, $p_s$ – the contained parts to the conjoin part – behaviours.

---

**Translation Schema 9**

\[
\text{value} \quad \text{BEHAVIOUR}_{\text{Part-P}arts \text{ Conjoin}(e)} \equiv \\
\quad \text{value} \\
\quad \quad \quad \quad M_C : \text{UI} \times \text{MT} \times \text{ST} \to \text{PT} \to \text{IOR} \quad \text{Unit} \\
\quad \quad \quad \quad M_C((ui,me,sv)(pv)) \equiv \beta_p((ui,me,sv)(pv))
\]

The next example focuses only on signatures.

---

**Road Transport Behaviour Signatures**

**Example 82** We first decide on names of behaviours. In the translation schemas we gave schematic names to behaviours of the form $M_p$. We now assign mnemonic names: from part names to names of transcendentally interpreted behaviours and then we assign signatures to these behaviours.

\[\text{value} \]

\[\text{hub}_{hui}:\]

\[a \quad \text{there is the usual “triplet” of arguments: unique identifier, mereology and static attributes;} \]
\[b \quad \text{then there are the programmable attributes;} \]
\[c \quad \text{and finally there are the input/output channel references: first those allowing communication between hub and link behaviours,} \]
\[d \quad \text{and then those allowing communication between hub and vehicle (bus and automobile) behaviours.} \]

\[\text{value} \]

\[\text{hub}_{hui}: \]
\[250 \quad \text{Hub}_{hui}: \text{UI} \times \text{vuis},\text{luis,}_\text{\{vuis,\text{luis}\} } \times \text{H\_Mer} \times \text{H\_\Omega} \]
\[250 \quad \quad \rightarrow \quad \text{(H\_\Sigma} \times \text{H\_Traffic}) \]
\[250 \quad \quad \text{in, out} \quad \{ \text{Hub}_{hui}\}_\text{\{vuis,\text{luis}\} } \quad \text{\{vuis,\text{luis}\}_\text{\{vuis,\text{luis}\} } \quad \text{Unit} \]
\[250 \quad \quad \text{pre: vuis } = \text{vuis} \wedge \text{luis } = \text{luis}^* \]

\[\text{link}_{hui}:\]

\[a \quad \text{there is the usual “triplet” of arguments: unique identifier, mereology and static attributes;} \]
\[b \quad \text{then there are the programmable attributes;} \]
6.7 Behaviour Signatures and Definitions

c and finally there are the input/output channel references: first those allowing communication between hub and link behaviours,
d and then those allowing communication between link and vehicle (bus and automobile) behaviours.

\begin{verbatim}
value
251 \textit{link}_i:
251 \textit{L}_{UI} \times (\text{vuis}, \text{huis} _) \times \text{L}_\text{Mer} \times \Omega
251 \textit{in} \rightarrow (\text{L}_\text{T} \times \text{L}_\text{Traffic})
251 \textit{out} \{ \text{h}_\text{ch} \text{h}_{ui}, \text{h}_{ui} | \text{h}_{ui} \text{h}_{ui} \text{h}_{ui} \text{h}_{ui} \in \text{huis} \}
251 \textit{pre} : \text{vuis} = v_{uis} \wedge \text{huis} = h_{uis}

252 \textit{bus}_i: \textit{bc}_{ui}:
252 \textit{bc}_{ui}: \text{BC}_{UI} \times (\text{buis}) \times \text{BC}_\text{Mer}
252 \textit{bus} \rightarrow \text{BusTimTbl}
252 \textit{in} \rightarrow \{ \text{bc}_{bc}_{ui} \text{bc}_{ui} | \text{bc}_{ui} \text{bc}_{ui} \text{bc}_{ui} \text{bc}_{ui} \in \text{buis} \}
252 \textit{pre} : \text{buis} = b_{uis} \wedge \text{huis} = h_{uis}

253 \textit{bus}_i: \textit{a}_{ui}:
253 \textit{a}_{ui}: \text{A}_{UI} \times (\text{bc}_{ui}, \text{ruis}) \times \text{A}_{Mer} \times \text{RegNo}
253 \textit{bus} \rightarrow \text{apo:APos}
253 \textit{in} \rightarrow \{ \text{ba}_{bc}_{ui} \text{r}_{ui} \text{a}_{ui} | \text{r}_{ui} \text{r}_{ui} \text{r}_{ui} \text{r}_{ui} \in \text{ruis} \}
253 \textit{pre} : \text{ruis} = r_{uis} \wedge \text{bc}_{ui} \in b_{uis}

254 \textit{automobile}_i:
254 \textit{a}_{ui}: \text{A}_{UI} \times (\text{ruis}) \times \text{A}_{Mer} \times \text{RegNo}
254 \textit{in} \rightarrow \text{apo:APos}
254 \textit{out} \{ \text{ba}_{bc}_{ui} \text{r}_{ui} \text{a}_{ui} | \text{r}_{ui} \text{r}_{ui} \text{r}_{ui} \text{r}_{ui} \in \text{ruis} \}
254 \textit{pre} : \text{ruis} = r_{uis} \wedge \text{a}_{ui} \in a_{uis}
\end{verbatim}

July 6, 2020, 10:05, A Foundation for Software Development
6.7.8 Core Behaviour

The core processes can be understood as never ending, “tail recursively defined” processes:

Core Behaviour Part e (I)

A Suggested Behaviour Definition 1

\[ R_F : UI \times MT \times ST \rightarrow PT \rightarrow IOR \rightarrow Unit \]

\[ R_F(ui, me, sv)(pv) \equiv \text{let } (me', pv') = R_F(ui, me, sv)(pv) \text{ in } R_F(ui, me', sv)(pa') \text{ end} \]

\[ F_F : UI \times MT \times ST \rightarrow PT \rightarrow IOR \rightarrow MT \times PT \]

We present a rough sketch of \( F_F \). The \( F_F \) action non-deterministically internal choice chooses between

- either \([1,2,3,4]\)
  - [1] accepting input from
  - [4] a suitable (“offering”) part process,
  - [2] optionally offering a reply, and
  - [3] finally delivering an updated state;
- or \([5,6,7,8]\)
  - [5] finding a suitable “order” (val)
  - [8] to a suitable (“inquiring”) behaviour (\( \pi' \)),
  - [6] offering that value (on channel ch[\( \pi' \)]),
  - [7] and then delivering an updated state;
- or [9] doing own work resulting in an updated state.

Core Behaviour Part e (II)

A Suggested Behaviour Definition 2

value

\[ F_F : UI \times MT \times ST \rightarrow PT \rightarrow IOR \rightarrow MT \times PT \]

\[ F_F(ui, me, sv)(pv) \equiv \]

\[
\begin{align*}
1 & \{ \text{let val } = \text{ch[\( \pi' \)]} ? \text{ in} \\
2 & ( \text{ch[\( \pi' \)]} ! \text{in_reply(val)(me,sv)(pv)} \mid \text{skip}) ; \\
3 & \text{in_update(val)(me,sv)(pv) end} \mid \pi': \text{II} \cdot \pi' \in \text{calc_iochnrefs(p)} \\
4 & \text{ch[\( \pi' \)]} ! \text{val} ; \\
5 & \text{out_update(val)(me,sv)(pv) end} \mid \pi': \text{II} \cdot \pi' \in \text{calc_iochnrefs(p)} \\
6 & \text{(me,own_work(sv)(pv))} \\
\end{align*}
\]

\[ \text{in_reply: VAL } \rightarrow \text{ST } \rightarrow \text{MT } \times \text{ST } \rightarrow \text{PT } \rightarrow \text{IOR } \rightarrow \text{VAL} \]

\[ \text{in_update: VAL } \rightarrow \text{MT } \times \text{ST } \rightarrow \text{PT } \rightarrow \text{IOR } \times \text{(MT } \times \text{PT)} \]

\[ \text{await_reply: UI } \rightarrow \text{MT } \times \text{ST } \rightarrow \text{PT } \rightarrow \text{IOR } \rightarrow \text{VAL} \]

\[ \text{out_update: VAL } \rightarrow \text{MT } \times \text{ST } \rightarrow \text{PT } \rightarrow \text{IOR } \rightarrow \text{MT } \times \text{PT} \]

\[ \text{own_work: SA } \rightarrow \text{MT } \times \text{PT } \rightarrow \text{IOR } \rightarrow \text{MT } \times \text{PT} \]

We leave these auxiliary functions and VAL undefined.

The in_reply, in_update, await_reply, out_update and own_work functions contain references to static and programmable attributes values by stating their names: sv and pv; and to monitorable attribute, \( A_m \), values by stating \( \text{attr}_A(part[ui]) \). Updates, \( v \), to biddable attributes, \( A_b \), are expressed as update(\( A,v,part[ui] \)).

Automobile Behaviour
6.7 Behaviour Signatures and Definitions

Example 83 We define the behaviours in a different order than the treatment of their signatures. We "split" definition of the automobile behaviour into the behaviour of automobiles when positioned at a hub, and into the behaviour automobiles when positioned at on a link. In both cases the behaviours include the "idling" of the automobile, i.e., its "not moving", standing still.

255 We abstract automobile behaviour at a Hub (hui).
256 The vehicle remains at that hub, "idling".
257 informing the hub behaviour,
258 or, internally non-deterministically,
   a moves onto a link, tl, whose "next" hub, identified by thui, is obtained from the mereology of the link identified by tlui;
   b informs the hub it is leaving and the link it is entering of its initial link position,
   c whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning (0) of that link,
259 or, again internally non-deterministically,
260 the vehicle "disappears — off the radar"!

255 automobile aui (aui,({},{ruis,vuis}),{},{ruis,vuis},rn)
256 (apos:atH(flui,hui,tlui)) \equiv
257 automobile aui (aui,({},{ruis,vuis}),{},{ruis,vuis},rn)(apos)
258 a (let ((fhui,thui,ruis)=mereo_\mathcal{L}(\varphi(tlui)) in
259 (assert: fhui=hui \land ruis=vuis)
260 (let onl = (tlui,hui,0,thui) in
261 (ba r ch[aui,hui] ! (record,TIME(),atH(flui,hui,tlui))) ||
262 (ba r ch[aui,tlui] ! (record,TIME(),onL(onl))) ||
263 automobile aui (aui,({},{ruis,vuis}),{},{ruis,vuis},rn)
264 (onL(onl)) end end)
265 stop

You may skip Example 84 in a first reading.

Further Behaviours of a Road Transport System

Example 84 Automobile Behaviour (on a link)

261 We abstract automobile behaviour on a Link.
   a Internally non-deterministically, either
      i the automobile remains, "idling", i.e., not moving, on the link,
      ii however, first informing the link of its position,
   b or
      i if if the automobile's position on the link has not yet reached the hub, then
         1 then the automobile moves an arbitrary small, positive Real-valued increment along the link
         2 informing the hub of this,
         3 while resuming being an automobile ate the new position, or
      ii else
         1 while obtaining a "next link" from the mereology of the hub (where that next link could very well be the same as the link the vehicle is about to leave),
         2 the vehicle informs both the link and the imminent hub that it is now at that hub, identified by thui,
         3 whereupon the vehicle resumes the vehicle behaviour positioned at that hub;
c or
d the vehicle “disappears — off the radar”!

\[\text{automobile}_{u}(a_{u},(\{}{}ruis,\{}{}\),rno)\]

\[\{\text{vp:onn}(\text{fh}_{u},\text{lu}_u,\text{th}_{u})\} =\]

\[\text{automobile}_{u}(a_{u},(\{}{}ruis,\{}{}\),rno)(\text{vp})\]

\[\text{if not}\ \text{atHub}(f)\]

\[\{\text{let incr = increment}(f)\ \text{in}\]

\[\text{let onl} = \text{atHub}(\text{fh}_{u},\text{th}_{u},\text{nx}_{u})\]

\[\text{automobile}_{u}(a_{u},(\{}{}ruis,\{}{}\),rno)(\text{vp})\]

\[\text{end} \}\]

\[\text{assert: m = atHub}(\text{fh}_{u},\text{th}_{u})\]

\[\text{hub}_{u}(h_{u}.(luis,vuis),h_{\omega})(h\sigma,ht)\equiv\]

\[\begin{align*}
\text{link behaviour non-deterministically, externally offers} \\
\text{to accept timed vehicle positions —} \\
\text{which will be on the link, from some vehicle, v}_{u}\).
\end{align*}\]

\[\begin{align*}
\text{timed vehicle link position is appended to the front of that vehicle’s entry in the link’s traffic table;}
\text{whereupon the link proceeds as a link behaviour with the updated link traffic table.}
\end{align*}\]

\[\begin{align*}
\text{the link behaviour offers to accept from any vehicle.}
\end{align*}\]

\[\begin{align*}
\text{A post condition expresses what is really a proof obligation: that the link traffic, lt' satisfies the axiom of the endurant link traffic attribute Item 169 Pg. 98.}
\end{align*}\]

\[\begin{align*}
\text{hub}_{u}(h_{u}.(luis,vuis),h_{\omega})(h\sigma,ht)\equiv
\end{align*}\]

\[\begin{align*}
\text{Link Behaviour} \quad \text{We model the hub behaviour vis-a-vis vehicles: buses and automobiles.}
\end{align*}\]
6.7 Behaviour Signatures and Definitions

We model bus companies very rudimentary. Bus companies keep a fleet of buses. Bus companies create, maintain, distribute bus time tables. Bus companies deploy their buses to honor obligations of their bus time tables. We shall basically only model the distribution of bus time tables to buses. We shall not cover other aspects of bus company management, etc.

Bus Company Behaviour

We model the interface between buses and their owning companies — as well as the interface between buses and the road net, the latter by almost “carbon-copying” all elements of the automobile behaviour(s).

272 The bus behaviour chooses to either

- accept a (latest) time-stamped bus time table from its bus company —
- where after it resumes being the bus behaviour now with the updated bus time table.

273 or, non-deterministically, internally,

- based on the bus position
  - if it is at a hub then it behaves as prescribed in the case of automobiles at a hub,
  - else, it is on a link, and then it behaves as prescribed in the case of automobiles on a link.

Bus Behaviour at a Hub

The atH\textsubscript{Bus}\textsubscript{bu} behaviour definition is a simple transcription of the automobile\textsubscript{a}\textsubscript{ui} (atH) behaviour definition: mereology expressions being changed from to, programmed attributes being changed from atH(fl\textsubscript{ui},h\textsubscript{ui},tl\textsubscript{ui}) to (ln,tt,atH(fl\textsubscript{ui},h\textsubscript{ui},tl\textsubscript{ui})), channel references a\textsubscript{ui} being replaced by b\textsubscript{ui}, and behaviour invocations renamed from automobile\textsubscript{a}\textsubscript{ui} to bus\textsubscript{bu}. So formula lines 256–261d below presents “nothing new”!

273(a)i atH\textsubscript{Bus}\textsubscript{bu}(b\textsubscript{ui},(\_,(bc\textsubscript{ui},ruis),\_))(ln,tt,bpos) ≡
273(a)i (ln,tt,atH(fl\textsubscript{ui},h\textsubscript{ui},tl\textsubscript{ui}))
6.8 System Initialisation

It is one thing to define the behaviours corresponding to all parts, whether composite or atomic. It is another thing to specify an initial configuration of behaviours, that is, those behaviours which “start” the overall system behaviour. The choice as to which parts, i.e., behaviours, are to represent an initial, i.e., a start system behaviour, cannot be “formalised”, it really depends on the “deeper purpose” of the system. In other words: requires careful analysis and is beyond the scope of the present monograph.

We sketch a general system initialisation function. It reflects the decision to transcendentally deduce all parts into behaviours.

value initialise_system: Unit → Unit
initialise_system() ≡
        let ps = calc_parts({uod}) in
6.8 System Initialisation

|| { let ui = uid_E(p), me = mereo_E(p),
sv = static_values(p), pv = programmable_values(p) in
  \( R_P(u_i, m_e, s_v)(p_v) \mid p : E \cdot p \in p s \)
end |

**Example 85 Initial States:** We recall the hub, link, bus company, bus and the automobile states outlined in Sect. 3.15 on Page 61.

value
110 \( hs : H - s e t \equiv \{ o b s _{SH}(o b s _{RN}(r t s)) \} \)
111 \( ls : L - s e t \equiv \{ o b s _{SL}(o b s _{RN}(r t s)) \} \)
113 \( bcs : B - s e t \equiv \{ o b s _{BC}(o b s _{FV}(o b s _{RN}(r t s))) \} \)
114 \( bs : B - s e t \equiv \{ o b_s (b) : b c = b c s \} \)
115 \( as : A - s e t \equiv \{ o b_s _{BC}(o b_s _{FV}(o b_s _{RN}(r t s))) \} \)

**Starting Initial Behaviours:** We are reaching the end of this domain modelling example. Behind us there are narratives and formalisations Item 117 Pg. 68– Item 260 Pg. 147. Based on these we now express the signature and the body of the definition of a “system build and execute” function.

275 The system to be initialised is

  a. the parallel compositions (\( || \)) of
  b. the distributed parallel composition (\( ||{\ldots}|| \)) of all the hub behaviours,
  c. the distributed parallel composition (\( ||{\ldots}|| \)) of all the link behaviours,
  d. the distributed parallel composition (\( ||{\ldots}|| \)) of all the bus company behaviours,
  e. the distributed parallel composition (\( ||{\ldots}|| \)) of all the bus behaviours, and
  f. the distributed parallel composition (\( ||{\ldots}|| \)) of all the automobile behaviours.

value
275 initial_system: Unit \( \rightarrow \) Unit
275 initial_system() \equiv
275b || { hub\(_{ui}(h\_ui, m\_e, h\_o)(htrf, h\_s)\) of all the hub behaviours, || h\( \in\) hs, h\_ui: H\_U\_H\_ui: uid\_H(h\_ui), me: H\_Met me: mereo\_H(h\_ui), \}
275b htrf: H\_Traffic htrf: attr\_H\_Traffic\_H(h\_ui),
275b h\_o: H\_\_o: attr\_H\_\_o(h\_ui), h\_s: H\_\_s: attr\_H\_\_s(h\_ui) \& h\_o \in h\_o }
275a ||
275c || { link\(_{ui}(l\_ui, l\_o)(ltrf, l\_s)\) of all the link behaviours, || l\_ui: L\_U\_L\_ui: uid\_L(l\_ui), me: L\_Met me: mereo\_L(l\_ui), \}
275c ltrf: L\_Traffic ltrf: attr\_L\_Traffic\_L(l\_ui),
275c l\_o: L\_\_o: attr\_L\_\_o(l\_ui), l\_s: L\_\_s: attr\_L\_\_s(l\_ui) \& l\_o \in l\_o }
275a ||
275d || { bus\(_{bc}(b\_bc, b\_bc)(btt)\) of all the bus company behaviours, || b\_bc: B\_C\_bc \in bcs, b\_bc\_ui: B\_C\_U\_b\_bc\_ui: uid\_B\_bc(b\_bc), me: B\_Met me: mereo\_B\_bc(b\_bc), \}
275d btt: Bus\_Tim\_T\_b\_tt: attr\_Bus\_Tim\_T\_b\_tt(b\_bc) }
275a ||
275e || { bus\(_{ui}(b\_ui, me)(ln, b\_ui)(bpos)\) of all the bus behaviours, || b\_ui: B\_U\_ui \in bcs, ln: L\_N: attr\_L\_N(b\_ui), \}
275e btt: Bus\_Tim\_T\_b\_tt: attr\_Bus\_Tim\_T\_b\_tt(b\_ui), bpos: B\_Pos b\_pos: attr\_B\_Pos(b\_ui) }
275a ||
275f || { automobile\(_{ai}(a\_ui, me, rn)(apos)\) of all the automobile behaviours, || a\_ui: A\_U\_a\_ui: uid\_A(a\_ui), me: A\_Met me: mereo\_A(a\_ui), \}
275f rn: Reg\_No\_n\_o: attr\_Reg\_No(a\_ui), apos: A\_Pos a\_apos: attr\_A\_Pos(a\_ui) }
6.9 Concurrency: Communication and Synchronisation

Translation Schemas 4, 7–9 reveal that two or more parts, which temporally coexist (i.e., at the same time), imply a notion of concurrency. Translation Schema 2 on Page 146, through the RSL/CSP language expressions $\text{ch}!v$ and $\text{ch}?$, indicates the notions of communication and synchronisation. Other than this we shall not cover these crucial notions related to parallelism.

6.10 Discrete Actions

In the extensive Road Transport System behaviour definitions, that is, in the Automobile Behaviour at a Hub Example 83 on Page 147, the Further Behaviours of a Road Transport (Appendix) Example 84 Page 147 and the Initial System example, Example 85 151, we have already “taken the lid off” the subject of action analysis & description, that is, unsystematically “revealed” aspects of action analysis & description. In this section we shall present a more systematic approach. We cannot do that for the full category of manifest domains such as we have defined domains. But we can single out a a sub-category of conjoin systems.

6.10.1 Conjoin Actions

The pragmatics of conjoins include (i) for single material conjoins, the transport, along the atomic part of the material, in one or both directions; (ii) for multiple, i.e., more than one material conjoins, the treatment of one or more of these materials, mixing, heating, “cleaning”, or other; as well as possibly also the transport of one or more of these materials to or from a conjoin, from or to “an outside”, and between conjoins.

Caveat: There is, however, a problem! The problem is that the domain phenomena that we really wish to model are not discrete in time, but continuous over time, and that we have no other means of modelling these phenomena that using good old-fashioned mathematical analysis, that is, partial differential equations as our analysis & description tool. Why is that a problem? It is a problem because we really have no “integrated” means of mixing the discrete mathematics-based notations – as here expressed in RSL – with that of classical mathematics’ partial differential equations, PDE, while making sure that the whole thing, the two notations, RSL and PDE, together makes sense, i.e., have a meaning. Research has gone on for now almost 30 years when this is written, but no real progress has been made. The discrete formal specification language research community, i.e., those of languages like, for example, VDM–SL, Z, RAISE, B, The B Method, et cetera, are naturally “steeped” in proof concerns where were and are not foremost in the minds of the PDE community. We refer to [10] for research papers on so-called “integrated formal methods”.

So, not choosing a problematic “mixture” of RSL and PDE we settle for just expressing some properties of actions on conjoin net parts. These are actions, to repeat, on parts but they involve materials and, although they are part actions they have consequences for “their materials”.

To express these operations we associate with conjoins just five simple operations: supply, pump, set valve, treat and dispose. These operations are operations “performed” by the part element of a conjoin, but they have more-or-less direct influence on the attributes of one or more of the material elements of a conjoin. There are no operations on forks, joins and pipes – what flows into these units flow out: is distributed, is collected and is just plainly forwarded. We shall therefore suggest an algebra of discrete operations. The inspiration for this algebra is derived from Yuri Gurevitch’s concept of evolving algebras, also now referred to as abstract state machines. We refer to [188, 103, 189, 104, 98, 103, 190, 97, 99, 104, 330, 331]. We refer to the operations that we shall suggest as discrete. That is, we shall not here consider these operations as “taking time”. We invite the reader to consider a temporal logic for domains while referring to [380, The Duration Calculus] and [263, Temporal Logic of Actions].

The conjoin operation make use of analyse_part_materials_part and analyse_part_materials_materials_endurant_analysis_function prompts defined in Sect. 3.17 on Page 63.
6.10 Discrete Actions

6.10.1.1 Discrete Supply of Material to Conjoins

A volume or weight amount of an appropriate substance is to be added to material \( m \) of endurant \( e \).

**Conjoin Operation 1**

Supply

\[
\begin{align*}
\text{let } p &: P &= \text{analyse_part_materials_part(e),} \\
& \quad (m:M) &= \text{analyse_part_materials_materials(e) in} \\
& \quad m' &:= \text{supply } m \text{ with } x(m^3|kg) \text{ of attr_Subs} \text{tance(m)} \\
\text{end}
\end{align*}
\]

The **Supply Schema** is to be understood as follows: A conjoin \( e \), a volume or weight amount, \( x \), and the material \( m \) of \( e \) is indicated. A specified amount, \( x \), of material is now added to that of \( m \) of \( e \) to become the new value, \( m' \), for that substance of \( e \). Typically \( e \) would be a supply unit of the material network, cf. Fig. 4.1 on Page 93.

6.10.1.2 Discrete Disposal of Material from Conjoins

A volume or weight amount, \( x \), of an appropriate substance is to be removed, i.e., disposed, from material \( m_i \) of endurant \( e \).

**Conjoin Operation 2**

Disposal

\[
\begin{align*}
\text{let } p &: P &= \text{analyse_part_materials_part(e),} \\
& \quad (m:M) &= \text{analyse_part_materials_materials(e) in} \\
& \quad m' &:= \text{dispose } x(m^3|kg) \text{ from } m \\
\text{end}
\end{align*}
\]

The **Disposal Schema** is to be understood as follows: A conjoin \( e \), a volume or weight amount, \( x \), of the material \( m \) of \( e \) is indicated. Somehow that amount of material is to be removed from that of \( m \) of \( e \) to become the new value, \( m' \), for that substance of \( e \).

6.10.1.3 Discrete Pumping of Material from Conjoins

We shall leave the interpretation of the following schemas, as a challenge, to the reader.

**Conjoin Operation 3**

Pump

\[
\begin{align*}
\text{let } p &: P &= \text{analyse_part_materials_part(e),} \\
& \quad (m:M) &= \text{analyse_part_materials_materials(e) in} \\
\text{let } (b,a) &= \text{before\_after\_conjoins(p)(),} \\
& \quad f &= \text{pumping\_capacity(p)} \text{ in} \\
\text{let } p,b &: PB &= \text{analyse_part_materials_part(b),} \\
& \quad (m_b:MB) &= \text{analyse_part_materials_materials(b),} \\
& \quad p,a &: PA &= \text{analyse_part_materials_part(a),} \\
& \quad (m_a:MA) &= \text{analyse_part_materials_materials(a) in} \\
& \quad m'_a &:= m_a \odot f(m_a); \\
& \quad m'_b &:= m_b \oplus f(m_a) \\
\text{end end end}
\end{align*}
\]
6.10.1.4 Discrete Opening/Closing of Material Transport by Valves

A valve is to be set at a fraction \( f \) of “flow-put” where \( 0 \leq f \leq 1 \).

**Conjoin Operation 4**

<table>
<thead>
<tr>
<th><strong>Valve</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>let ( p : P ) = analyse_part_materials_part(e), ( (m_\ldots) = \text{analyse_part_materials_materials}(e) ) in</td>
</tr>
<tr>
<td>( \triangleright \ m' := \text{set valve opening at } f \text{ for material } m )</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

6.10.1.5 Discrete Treatment of Materials of a Conjoin

**Conjoin Operation 5**

<table>
<thead>
<tr>
<th><strong>Treatment</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>let ( p : P ) = analyse_part_materials_part(e), ( (m_1,m_2,\ldots,m_m) = \text{analyse_part_materials_materials}(e) ) in axiom ( m \geq 1 )</td>
</tr>
<tr>
<td>( \triangleright \ m'_1 := \text{treat with } a_1/b \text{ of } m_1, c_1/d \text{ of } m_2, \ldots, e_1/f \text{ of } m_m \text{ with operation } o_1, )</td>
</tr>
<tr>
<td>( \triangleright \ m'_2 := \text{treat with } a_2/b \text{ of } m_1, c_2/d \text{ of } m_2, \ldots, e_2/f \text{ of } m_m \text{ with operation } o_2, )</td>
</tr>
<tr>
<td>( \triangleright \ldots )</td>
</tr>
<tr>
<td>( \triangleright \ m'_m := \text{treat with } a_m/b \text{ of } m_1, c_m/d \text{ of } m_2, \ldots, e_m/f \text{ of } m_m \text{ with operation } o_m )</td>
</tr>
<tr>
<td>axiom ( \forall i : \text{Nat} \cdot 1 \leq i \leq m \Rightarrow a_i \leq b \land c_i \leq d \land \ldots \land e_i \leq f \land )</td>
</tr>
<tr>
<td>( \land \forall (x_i,y) : {(a_i,b),(c_i,d),\ldots,(e_i,f)} \cdot x_1 + x_2 + \ldots + x_m = y )</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

If, for some \( i \), \( m'_i \) is to have no contribution from some \( m_j \) then \( x_j/y = 0 \), i.e., \( x_j = 0 \).

The **Treatment Schema** is to be understood as follows: There are up to \( m \) ‘assignments’. They are to be understood as an equation system. The [to the] right [of :=] \( m_i's \) all have fixed, initial values. The [to the] left [of :=] \( m'_i \) denote a final value. By value we mean that either, for all entries of an ‘assignment/equation’, we speak of **Volume**, or we speak of **Weight**. Et cetera!

\[ \cdots \]

You may think of the above \( \triangleright \)'s to single out the actual operations on conjoin parts. The schema text surrounding these \( \triangleright \) lines serve to identify the quantities involved in the operations. So the conjoin part actions are, in a sense, loosely described. We refer to Exercise 27 on Page 159.

\[ \cdots \]

This section ends a series of discourses on conjoins. We refer to Sect. 3.18.4 on Page 69, Example 40 on Page 71, Sect. 4.3.7 on Page 93 and Sect. 4.4.8 on Page 108. This last “installment” has been, but a sketch. We refer to Sect. 7.4 on Page 171 on **Rules & Regulations**, Sect. 7.5 on Page 173 on **Scripts** and Sect. 7.6 on Page 175 on **License Languages**, Pages 171–184, for a continuation of the subject.

6.11 Discrete Events

To clear any possible misunderstanding there are two kinds of events. There are the domain events that we shall analyse & describe; and there are the events of the domain description. The latter are exemplified by CSP’s out/input clauses: \( \text{ch}[..] ! e \) (offer value of expression \( e \) on channel \( \text{ch}[..] \)), and \( \text{ch}[..] ? (accept value offered on channel \( \text{ch}[..] \)). We shall use the latter to model the former!
By domain event we shall understand a change of domain state for which we do, or cannot, point out a known domain behaviour to be the cause of that event.

### Domain Events

**Example 86** We informally sketch some domain events. (i) An automobile suddenly skidding off a link or hub, thus, in sense, “disappearing” from the road net, rendering the transport domain in chaos – if we are not prepared to model the recovery, as is done in the domain, from such calamities. (ii) A pipeline unit suddenly bursting, i.e., exploding, thus, in a sense, rendering the pipeline in chaos – if we are not prepared to model the recovery, as is done in the domain, from such calamities.

We suggest to model domain events as follows. Let

$$B_P: UI \times MT \times ST \to PT \to IOT \text { Unit}$$

$$B_P(ui,me,sv)(pv) \equiv \text{let } (me',pv') = F_P(ui,me,sv)(pv) \text{ in } P_P(ui,me',sv)(pv') \text{ end}$$

be the body of a behaviour definition. Domain events that can be, say approximately, identified as taking place in a resumption of $$P_P(ui,me',sv)(pa')$$ can then be expressed in a changed definition of $$B_P$$:

**value**

$$B_P: UI \times MT \times ST \to PT \to IOT \text { Unit}$$

$$B_P(ui,me,sv)(pv) \equiv$$

(1.) \text{let } (me',pv') = F_P(ui,me,sv)(pv) \text{ in }$

(2.) \text{either: chaos}$

(3.) \text{or: let } (me'',sv',pv'') = \text{handle_event}_{P} (me',sv,pv') \text{ in } P_P(ui,me'',sv')(pv'') \text{ end}$

(4.) $$\lceil \rceil P_P(ui,me',sv)(pv) \text{ end}$$

\text{handle_event: } MT \times ST \times PT \to MT \times ST \times PT$$

We informally explain: (1.) is as for the “un-event” version of $$B_P$$. Then modelling the occurrence of possibly not-occurring events means that the behaviour non-deterministically, line (4.), chooses the (2.-3.) model or the (4.) model. (2.) \text{either} the domain analyser & describer chooses to not handle event handle_event, and specifies chaos; (3.) \text{or} the domain analyser & describer chooses to model some handling of the event – before resuming $$P_P$$. (4.) In this model no event has been “detected” – and life proceeds as normal.

Similar domain events occurring “during” $$F_P$$ can be handled likewise.

### 6.12 A Domain Discovery Process, III

The predecessors of this section are Sects. 3.20 on Page 74 and 4.8 on Page 116.

We shall yet again emphasize some aspects of the domain analyser & describer method. A method principle is that of exhaustively analyse & describe all external qualities of the domain under scrutiny. A method technique implied here is that sketched below. The method tools are here all the analysis and description prompts covered so far.

#### 6.12.1 Review of The Endurant Analysis and Description Process

The endurant analysis & description process is defined in Sect. 4.8 on Page 116.

**value**

$$\text{endurant_analysis_and_description: Unit } \to \text{ Unit}$$

$$\text{endurant_analysis_and_description()} \equiv$$

- discover_sorts();
- discover_uids();
- discover_mereologies();
- discover_attributes();
We are now to define a perdurant_analysis_and_description procedure – to follow the above endurant_analysis_and_description procedure.

6.12.2 A Perdurant Analysis and Description Process

We define the perdurant_analysis_and_description procedure in the reverse order of that of Sect. 4.8 on Page 116, first the full procedure, then its sub-procedures.

<table>
<thead>
<tr>
<th>A Domain Endurant Analysis and Description Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
</tr>
<tr>
<td>perdurant_analysis_and_description: Unit → Unit</td>
</tr>
<tr>
<td>perdurant_analysis_and_description() ≡</td>
</tr>
<tr>
<td>discover_state(); axiom [ Note (a) ]</td>
</tr>
<tr>
<td>discover_channels(); axiom [ Note (b) ]</td>
</tr>
<tr>
<td>discover_behaviour_signatures(); axiom [ Note (c) ]</td>
</tr>
<tr>
<td>discover_behaviour_definitions(); axiom [ Note (c) ]</td>
</tr>
<tr>
<td>discover_initial_system() axiom [ Note (d) ]</td>
</tr>
</tbody>
</table>

Note (a) The State: The state variable parts maps unique identifiers of every part into that part. We might, perhaps should, modify “that part” into a quantity to which monitorable attribute value inquiries, attr_A, apply; and nothing more, that is, “parts” devoid themselves of unique identifiers, mereology, and static and programmable attributes. We refrain from doing so here.

Note (b) The Channels: We refer to Sect. 6.5.2 on Page 134. Thus we indiscriminately declare a channel for each pair of distinct unique part identifiers whether the corresponding pair of part behaviours, if at all invoked, communicate or not.

Note (c) Discrete Behaviour Signatures and Definitions: In Sect. 6.7 on Page 140 Translation Schemas 3–9 “lump” expression of behaviour signature and definition into one RSL-Text. Here we separate the two. The reason is one of pragmatics. We find it more productive to first settle on the signatures of all behaviours before tackling the far more time-consuming work on defining the behaviours.

Note (d) The Running System: We refer to Sect. 6.8 on Page 150.

6.12.2.1 The discover_state Procedure

We model the state of all parts as a globally declared variable parts, which is modelled as a map from the unique identifiers of parts to their [initial] value, that is, parts(ui). We need basically only model those parts, p, which have monitorable attributes, say A, as their values, attr_A(p), need be read, that is attr_A(parts(ui)).

| value                                             |
| discover_state: Unit → Unit                       |
| discover_state() ≡                               |
| for ∨ v · v ∈ gen do                             |
| txt := txt ↑ [type_name(v)→txt(type_name(v))→(describe_state(v)) ] end |

| describe_state: E → RSL-Text                     |
| describe_state(e) ≡ ※ variable parts[uid,E(e)];type_name(e) := e ※ |

6.12.2.2 The discover_channels Procedure

We refer to Sects. 4.2.4 on Page 87 and 6.5.2 on Page 134.

| value                                             |
| discover_channels: Unit → Unit                    |
| discover_channels() ≡                            |
let ch_text = channel \{ ch[i,j,k] | i,j,k: UI \land i \neq j \land i \neq k \land \text{all uniq ids()} : M \} in

\text{txt} := \text{txt} \uparrow [\text{type name(uod)} \mapsto (\text{ch_text}) \uparrow \text{txt}(\text{type name(uod)})]

end

6.12.2.3 The discover_signatures Procedure

We refer to Sect. 6.7 on Page 140.

\text{value}
\text{discover\_behaviour\_signatures: Unit \rightarrow Unit}
\text{discover\_behaviour\_signatures()} \equiv
\text{for } \forall v \in \text{gen} \text{ do}
\text{let signature } =
  \text{is\_atomic}(v) \rightarrow \langle \text{MP : UI } \times \text{MT } \times \text{ST } \rightarrow \text{PT } \rightarrow \text{IOR Unit} \rangle,
  \text{is\_composite}(v) \rightarrow \langle \text{ME : UI } \times \text{MT } \times \text{ST } \rightarrow \text{PT } \rightarrow \text{IOR Unit} \rangle,
  \text{is\_structure}(v) \rightarrow \langle \rangle,
  \text{is\_part\_materials\_conjoin}(v) \rightarrow
    \langle \text{MC : UI } \times \text{MT } \times \text{ST } \rightarrow \text{PT } \times (\text{MT }_1 \times \text{MT }_2 \times \ldots \times \text{MT }_m) \rightarrow \text{IOR Unit} \rangle,
  \text{is\_material\_parts\_conjoin}(v) \rightarrow \langle \text{MC : UI } \times \text{MT } \times \text{ST } \rightarrow \text{PT } \times \text{CM} \rightarrow \text{IOR Unit} \rangle,
  \text{is\_part\_parts\_conjoin}(v) \rightarrow \langle \text{MC : UI } \times \text{MT } \times \text{ST } \rightarrow \text{IOR Unit} \rangle \text{ in }
\text{txt} := \text{txt} \uparrow [\text{type name(v)} \mapsto \text{txt}(\text{type name(v)})\uparrow \text{signature}]

end end

6.12.2.4 The discover\_behaviour\_definitions Procedure

We refer to Sect. 6.7 on Page 140.

\text{value}
\text{discover\_behaviour\_definitions: Unit } \rightarrow \text{Unit}
\text{discover\_behaviour\_definitions()} \equiv
\text{for } \forall v \in \text{gen} \text{ do}
\text{let definition } =
  \text{is\_atomic}(v) \rightarrow \langle \text{MP}(ui,me,sv)(pv) \equiv \text{RP}(ui,me,sv)(pv) \rangle,
  \text{is\_composite}(v) \rightarrow \langle \text{ME}(ui,me,sv)(pv) \equiv \text{RP}(ui,me,sv)(pv) \rangle,
  \text{is\_structure}(v) \rightarrow \langle \rangle,
  \text{is\_part\_materials\_conjoin}(v) \rightarrow \langle \text{MC}(ui,me,sv)(pv,cm) \equiv \text{RP}(ui,me,sv)(pv,cm) \rangle,
  \text{is\_material\_parts\_conjoin}(v) \rightarrow \langle \text{MC}(ui,me,sv)(pv,cm) \equiv \text{RP}(ui,me,sv)(pv,cm) \rangle,
  \text{is\_part\_parts\_conjoin}(v) \rightarrow \langle \text{MC}(ui,me,sv)(pv) \equiv \text{RP}(ui,me,sv)(pv) \rangle \text{ in }
\text{txt} := \text{txt} \uparrow [\text{type name(v)} \mapsto \text{txt}(\text{type name(v)})\uparrow \text{definition}]

end end

6.12.2.5 The initialise\_system Procedure

We refer to Sect. 6.8 on Page 150 (for initialise\_system()).

\text{value}
\text{discover\_initial\_system: Unit } \rightarrow \text{Unit}
\text{discover\_initial\_system()} \equiv \text{txt} := \text{txt} \uparrow [\text{UoD } \mapsto \text{txt}(\text{UoD}) \uparrow \langle \text{initialise\_system}() \rangle]
6.13 Summary

This chapter's main title was: DOMAINS – Towards a Dynamics Ontology. The term 'Dynamics' pertain to actions, events and behaviours of the 'Domain', not to its 'Ontology'. So, an aspect of the ontology of a domain, such as we have studied it and such as we ordain one aspect of domain analysis & description, is also about the time-evolving occurrence, of actions, events and behaviours, that is, the perdurants. For that study & practice we have suggested a number of analysis & description prompts.

6.13.1 Method Principles, Techniques and Tools

Recall that by a method we shall understand a set of principles for selecting and applying a set of techniques using a set of tools in order to construct an artefact.

6.13.1.1 Principles of Perdurant Analysis & Description

In this chapter we have illustrated the use of the following principles:

**Divide & Conquer**: That concept is addressed in the sequential treatment of states, channels, behaviour signatures and behaviour definitions.

**Operational Abstraction**: That concept is addressed in several ways: in the formulation of a notion of domain states and its modelling in terms of RSL variables; in the modelling of domain behaviour interactions in terms of CSP channels, output and input; in the capturing of one essence of domain behaviours in terms of the signature of CSP process definitions; and in the modelling of domain behaviours in terms of CSP processes.

In this chapter we have put forward some advice on description choices: We refer to Modelling Choices 7 on Page 142 and 8 on Page 143. The analysis predicates and functions are merely aids. They do not effect descriptions, but descriptions are based on the result of their inquiry. Real decisions are made when effecting a description function. So the rôle of these modelling choice paragraphs is to alert the describer to make judicious choices.

6.13.1.2 Techniques of Perdurant Analysis & Description

In this chapter we have illustrated the use of the following techniques:

- **Modelling Behaviours as CSP Processes**: With that choice follows then the “standard” CSP techniques of tail-recursive specification of concurrent processes/behaviours [237, 239, 238, 336, 340].
- **Interpretation of Internal Endurant Qualities**: As part of the translation of endurant parts to CSP processes follows the interpretation of unique part identifiers as constant process identifiers; part mereologies as determinant for CSP communication channel indices; static part attributes as constant process, “by value” arguments; programmable part attributes as such process arguments that can be given new values when tail-recursively [re-]invoked; and monitorable part attributes as “residing” in RSL declared variables, one for each part having monitorable attributes.

6.13.1.3 Tools of Perdurant Analysis & Description

In this chapter we have illustrated the use of the following tools:

6.13.1.3.1 Analysis Functions

- `calc_allchndcls` Item 243a on Page 139
- `calcchnrefs` Item 242a on Page 139
- `calcchnrefsm` Item 241 on Page 139
- `moniattrtypes` Item 183 on Page 100
- `moniattrvals` Item 186 on Page 101
6.14 Exercise Problems

6.14.1 Research Problems

Exercise 27 A Research Challenge. PED Specification of Flows in Oil Pipelines: We refer to Example 40 on Page 71, Sect. 6.10.1 on Page 152, Appendix Sect. A, and to Exercise 28 on the following page. This research problem addresses an open problem. You are to assume a domain of oil pipelines. In this monograph many examples and term project exercises show fragments of, respectively are intended to develop, a full domain description of road transport systems. Now you have in Appendix A a rather complete description of the endurants of an oil pipeline domain.

The problem to be studied, and for which we seek partial domain descriptions in some form of classical, and, perhaps, not so classical mathematics: partial differential equations, PDE, is the fluid dynamics of the flow of oil in pipelines – and, for that matter, in any net of part-fluid-material conjoins.

- [Q1] First we ask you to set of the fluid dynamics for each kind of pipeline units: well, pump, pipe, valve, fork and sink.
- [Q2] Then we ask that you, as a “little, preparatory exercise”, “glue” the fluid dynamics, i.e., their mathematical equations, of pairs of pipeline units:

$$\infty (\text{well};\text{pump}), \infty (\text{pipe};\text{valve}), \infty (\text{fork};(\text{pipe}|\text{pipe}))^{15}, \infty (\text{pipe};\text{sink}).$$

$$\infty (\text{pump};\text{pipe}), \infty (\text{valve};\text{pipe}), \infty ((\text{pipe}|\text{pipe});\text{join})^{16},$$

$$\infty (\text{pipe};\text{pipe}), \infty (\text{pipe};\text{fork}), \infty (\text{join};\text{pipe}) \text{ and}$$

---

15 – the | operator in (pipe|pipe) is intended to informally express that two pipes “emanate” a fork
16 – the | operator in (pipe|pipe) is intended to informally express that two pipes “enter” a join
[Q3] Finally we ask you to consider the fluid dynamics of an entire pipeline system. That is, for any pipeline system we seek a definite set of possibly somehow “parameterised” definite sets of \( \text{Peds} \), or whatever mathematics it takes, to model the full dynamics of any one such pipeline system!

- Make suitable assumptions.
- [Q4] Publish the result!
- [Q5] Then start thinking about how to “blend” the \( \text{PDEs} \) into an \( \text{RSL} \) specification. What might we mean by ‘blend’?

### 6.14.2 Student Exercises

#### Exercise 28 A PhD Student Problem. RSL Specification of Flows in Oil Pipelines

We refer to Sect. 6.10.1 on Page 152. It is now suggested that you, in turn, one-by-one, consider the following sub-problems in the context of your domain of conjugate endurants be that of a ocean, canal and harbour (basin) system with cargo liners.

- Define the external qualities of a domain of shipping lines based on the hinted waterways and vessels – and based on there being shipping lines and terminal port that own, , keep track of operate and service (load, unload) the cargo liner vessels.
- Define relevant internal qualities, one-by-one:
  - unique identifiers,
  - mereologies and,
  - a small set of attributes.
- Now single one or two conjugates and cargo lines and terminal ports out for consideration as behaviours. Suggest, in turn,
  - an overall state,
  - a set of domain model channels,
  - signatures of behaviours, and
  - definition of behaviours.

#### Exercise 29 An MSc Student Exercise. Document System Actions

We refer to Exercises 3 on Page 80, 16 on Page 120, 17 on Page 120 and 18 on Page 120. We also refer to Sect. 6.10.1 on Page 152.

- In line with the \textit{conjoin operation schemas}, shown in Sect. 6.10.1, you are to provide \textit{document operation schemas} for the operations mentioned in Exercises 3, 16, 17 and 18. We refer to Exercise 30.

#### Exercise 30 An MSc Student Exercise. Document System Behaviours

We assume and refer to exercise 29. You are to narrate and formalise the full set of document system perdurants: the channels, cf. Sect. 6.5 on Page 133 and Example 80 on Page 135 the behaviour signatures, cf. Sect. 6.6 on Page 136 and Example 82 on Page 144, the behaviour definitions, cf. Sect. 6.6 on Page 136 and Example 82 on Page 144, and the initial, i.e., the ‘running’ document system, i.e., the “start-up”, cf. Sect. 6.8 on Page 150 and Example 85 on Page 151.

17 Any such pipeline system has its “root” in, say, the formulas of Example 40 on Page 71 and Appendix A where we present a rather comprehensive description of the endurants of a pipeline system.


19 Examples of assumptions are that there are defined all the necessary attributes concerning pipeline units’ liquid flow properties.

20 A standard concern is that of being able to carry out either mathematical logic proofs (of properties) or conventional mathematical reasoning over ‘blended’, say, RSL expressed models and classic mathematical models.
6.14.3 Term Projects

We continue the term projects of Sects. 3.24.3 on Page 81 and 4.12.3 on Page 121.

For the specific domain topic that a group is working on it is to treat, for example, in separate, typically four, consecutive weeks, these topics in the order listed:

- *Channels*, cf. Sect. 6.5 on Page 133 and Example 80 on Page 135;
- *Behaviour Signatures*, cf. Sect. 6.6 on Page 136 and Example 82 on Page 144;
- *Discrete Behaviour Definitions*, cf. Sect. 6.3.4 on Page 127, cf. Example 84 on Page 147;

**Exercise 31** An MSc Student Exercise. The Consumer Market, Perdurants: We refer to Exercises 4 on Page 82 and 20 on Page 121.

**Exercise 32** An MSc Student Exercise. Financial Service Industry, Perdurants: We refer to Exercises 5 on Page 82 and 21 on Page 121.

**Exercise 33** An MSc Student Exercise. Container Line Industry, Perdurants: We refer to Exercises 6 on Page 82 and 22 on Page 121.

**Exercise 34** An MSc Student Exercise. Railway Systems, Perdurants: We refer to Exercises 7 on Page 82 and 23 on Page 121.

**Exercise 35** A PhD Student Problem. Part-Material Conjoins: Canals, Perdurants: We refer to Exercises 8 on Page 82 and 24 on Page 121.

**Exercise 36** A PhD Student Problem. Part-Materials Conjoins: Rum Production, Perdurants: We refer to Exercises 9 on Page 82 and 25 on Page 121.

**Exercise 37** A PhD Student Problem. Part-Materials Conjoins: Waste Management, Perdurants: We refer to Exercises 10 on Page 82 and 26 on Page 122.

These exercise problems are continued in Sects. 7.11.2 on Page 193 and 8.9.2 on Page 241.
7

DOMAIN FACETS

In this chapter we introduce the concept of domain facets. We cover the following facets: intrinsics, support technologies, rules and regulations, scripts, license languages, management & organisation and human behaviour.

7.1 Introduction

In Chapters 3–6 we outlined a method for analysing and describing domains. In this chapter we cover some domain analysis & description principles and techniques not covered in Chapters 3–6. Those chapters focused on manifest domains. Here we, on one side, go “outside” the realm of manifest domains, and, on the other side, cover, what we shall refer to as, facets, not covered in Chapters 3–6.

7.1.1 Facets of Domains

By a domain facet we shall understand one amongst a finite set of generic ways of analysing a domain: a view of the domain, such that the different facets cover conceptually different views, and such that these views together cover the domain. Now, the definition of what a domain facet is can seem vague. It cannot be otherwise. The definition is sharpened by the definitions of the specific facets. You can say, that the definition of domain facet is the “sum” of the definitions of these specific facets. The specific facets – so far – are:

- intrinsics (Sect. 7.2),
- support technology (Sect. 7.3),
- rules & regulations (Sect. 7.4),
- scripts (Sect. 7.5),
- license languages (Sect. 7.6),
- management & organisation (Sect. 7.7) and
- human behaviour (Sect. 7.8).

Of these, the rules & regulations, scripts and license languages are closely related. Vagueness may “pop up”, here and there, in the delineation of facets. It is necessarily so. We are not in a domain of computer science, let alone mathematics, where we can just define ourselves precisely out of any vagueness problems. We are in the domain of (usually) really world facts. And these are often hard to encircle.

7.1.2 Relation to Previous Work

The present chapter is a rather complete rewrite of [52]. The reason for the rewriting was the expected publication of [78]. [52] was finalised already in 2006, 10 years ago, before the analysis & description calculus of [78] had emerged. It was time to revise [52] rather substantially.

1 We write: ‘so far’ in order to “announce”, or hint that there may be other specific facets. The one listed are the ones we have been able to “isolate”, to identify, in the most recent 10-12 years.
7.1.3 Structure of Chapter

The structure of this chapter follows the seven specific facets, as listed above. Each section, 7.2.–7.8., starts by a definition of that specific facet. Then follows an analysis of the abstract concepts involved usually with one or more examples – with these examples making up most of the section. We then “speculate” on derivable requirements thus relating the present chapter to [66]. We close each of the sections, 7.2.–7.8., with some comments on how to model the specific facet of that section.

Examples 87–108 of sections 7.2–7.8 present quite a variety. In that, they reflect the wide spectrum of facets.

More generally, domains can be characterised by intrinsically being endurant, or function, or event, or behaviour intensive. Software support for activities in such domains then typically amount to database systems, computation-bound systems, real-time embedded systems, respectively distributed process monitoring and control systems. Other than this brief discourse we shall not cover the “intensity”-aspect of domains in this chapter.

7.2 Intrinsics

• By domain intrinsics we shall understand those phenomena and concepts of a domain which are basic to any of the other facets (listed earlier and treated, in some detail, below), with such domain intrinsics initially covering at least one specific, hence named, stakeholder view

7.2.1 Conceptual Analysis

The principles and techniques of domain analysis & description, as unfolded in Chapters 3–6, focused on and resulted in descriptions of the intrinsics of domains. They did so in focusing the analysis (and hence the description) on the basic endurants and their related perdurants, that is, on those parts that most readily present themselves for observation, analysis & description.

---

**Railway Net Intrinsics**

**Example 87** We narrate and formalise three railway net intrinsics. From the view of potential train passengers a railway net consists of lines, l:L, with names, Ln:Ln, stations, s:S, with names sn:Sn, and trains, tn:TN, with names tnm:Tnm. A line connects exactly two distinct stations.

```plaintext
scheme N0 =
  class
    type
      N, L, S, Sn, Ln, TN, Tnm
    value
      obs_Ls: N → L-set, obs_Ss: N → S-set
      obs_Ln: L → Ln, obs_Sn: S → Sn
      obs_Sns: L → Sn-set, obs_Lns: S → Ln-set
    axiom
      ... end

N, L, S, Sn and Ln designate nets, lines, stations, station names and line names. One can observe lines and stations from nets, line and station names from lines and stations, pair sets of station names from lines, and lines names (of lines) into and out from a station from stations. Axioms ensure proper graph properties of these concepts.
```
From the view of actual train passengers a railway net — in addition to the above — allows for several lines between any pair of stations and, within stations, provides for one or more platform tracks, tr: Tr, with names, trn:Trn, from which to embark on or alight from a train.

\[
\text{scheme N1 = extend N0 with class type } \begin{align*}
\text{Tr, Trn} \\
\text{value obs_Trs: S \rightarrow Tr-set, obs_Trn: Tr \rightarrow Trn} \\
\text{axiom} \\
\ldots
\end{align*}
\]

The only additions are that of track and track name types, related observer functions and axioms.

From the view of train operating staff a railway net — in addition to the above — has lines and stations consisting of suitably connected rail units. A rail unit is either a simple (i.e., linear, straight) unit, or is a switch unit, or is a simple crossover unit, or is a switchable crossover unit, etc. Simple units have two connectors. Switch units have three connectors. Simple and switchable crossover units have four connectors. A path, p:P, (through a unit) is a pair of connectors of that unit. A state, σ: Σ, of a unit is the set of paths, in the direction of which a train may travel. A (current) state may be empty: The unit is closed for traffic. A unit can be in any one of a number of states of its state space, ω: Ω.

\[
\text{scheme N2 = extend N1 with class type } \begin{align*}
\text{U, C} \\
\text{P} &= U \times (C \times C) \\
\text{P} &= \{ (u,(c,c')) \mid p \text{ in } (c,c') \in \bigcup \text{obs}_\Omega(u) \} \\
\Sigma &= P\text{-set} \\
\Omega &= \Sigma\text{-set} \\
\text{value obs_USART: (N|L|S) \rightarrow U-set} \\
\text{obs_USC: U \rightarrow C-set} \\
\text{obs_US: U \rightarrow Σ} \\
\text{obs_UΩ: U \rightarrow Ω} \\
\text{axiom} \\
\ldots
\end{align*}
\]

Unit and connector types have been added as have concrete types for paths, unit states, unit state spaces and related observer functions, including unit state and unit state space observers.

Different stakeholder perspectives, not only of intrinsics, as here, but of any facet, lead to a number of different models. The name of a phenomenon of one perspective, that is, of one model, may coincide with the name of a “similar” phenomenon of another perspective, that is, of another model, and so on. If the intention is that the “same” names cover comparable phenomena, then the developer must state the comparison relation.

**Intrinsics of Switches**

**Example 88** The intrinsic attribute of a rail switch is that it can take on a number of states. A simple switch (Y) has three connectors: \{c, c_i, c_j\}. c is the connector of the common rail from which one can
either "go straight" $c_1$, or "fork" $c_f$ (Fig. 7.1). So we have that a possible state space of such a switch

$$\omega_g = \{\emptyset, \{(c, c_1)\}, \{(c_1, c)\}, \{(c, c_1), (c_1, c)\}, \{(c, c_1), (c_1, c), (c, c_f), (c_f, c)\}, \{(c, c_1), (c_1, c), (c, c_f), (c_f, c), (c, c_f), (c_f, c)\}\}

The above models a general switch ideally. Any particular switch $\omega_p$ may have $\omega_p \subseteq \omega_g$. Nothing is said about how a state is determined: who sets and resets it, whether determined solely by the physical position of the switch gear, or also by visible or virtual (i.e., invisible, intangible) signals up or down the rail, away from the switch.

![Fig. 7.1. Possible states of a rail switch](image-url)

**Example 89** Think of documents, written, by hand, or typed "onto" a computer text processing system.

One way of considering such documents is as follows. First we abstract from the syntax that such a document, or set of more-or-less related documents, or just documents, may have: whether they are letters, with sender and receive addressees, dates written, sent and/or received, opening and closing paragraphs, etc., etc.; or they are books, technical, scientific, novels, or otherwise, or they are application forms, tax returns, patient medical records, or otherwise. Then we focus on the operations that one may perform on documents: their creation, editing, reading, copying, authorisation, "transfer"\(^2\), "freezing"\(^3\), and shredding. Finally we consider documents as manifest parts, cf. Chapter 3, Parts, so documents have unique identifications, in this case, changeable mereology, and a number of attributes. The mereology of a document, $d$, reflects those other documents upon which a document is based, i.e., refers to, and/or refers to $d$. Among the attributes of a document we can think of (i) a trace of what has happened to a document, i.e., a trace of all the operations performed on "that" document, since and including creation — with that trace, for example, consisting of time-stamped triples of the essence of the operations, the "actor" of the operation (i.e., the operator), and possibly some abstraction of the locale of the document when operated upon; (ii) a synopsis of what the document text “is all about”; (iii) and some “rendition” of the document text. We refer to experimental technical research report [71].

This view of documents, whether “implementable” or “implemented” or not, is at the basis of our view of license languages (for digital media, health-care (patient medical record), documents, and transport (contracts) as that facet is covered in Sect. 7.6.
7.2.2 Requirements

Chapter 8 illustrates requirements “derived” from the intrinsics of a road transport system – as outlined in Chapters 3–6. So the present chapter has little to add to the subject of requirements “derived” from intrinsics.

7.2.3 On Modeling Intrinsics

Chapters 3–6 outline basic principles, techniques and tools for modeling the intrinsics of manifest domains. Modeling the domain intrinsics can often be expressed in property-oriented specification languages (like CafeOBJ [157]), model-oriented specification languages (like Alloy [251], B [1], VDM-SL [88, 89, 154], RSL [176], or Z [374]), event-based languages (like Petri nets or [332] or CSP [238]), respectively in process-based specification languages (like MSCs [249], LSCs [200], Statecharts [199], or CSP [238]). An area not well-developed is that of modeling continuous domain phenomena like the dynamics of automobile, train and aircraft movements, flow in pipelines, etc. We refer to [302].

7.3 Support Technologies

- **By a domain support technology we shall understand ways and means of implementing certain observed phenomena or certain conceived concepts.**

The “ways and means” may be in the form of “soft technologies”: human manpower, see, however, Sect. 7.8, or in the form of “hard” technologies: electro-mechanics, etc. The term ‘implementing’ is crucial. It is here used in the sense that, \( \psi \tau \), which is an ‘implementation’ of a endurant or perdurant, \( \phi \), is an extension of \( \phi \), with \( \phi \) being an abstraction of \( \psi \tau \). We strive for the extensions to be proof theoretic conservative extensions [274].

7.3.1 Conceptual Analysis

There are [always] basically two approaches the task of analysing & describing the support technology facets of a domain. One either stumbles over it, or one tries to tackle the issue systematically. The “stumbling” approach occurs when one, in the midst of analysing & describing a domain realises that one is tackling something that satisfies the definition of a support technology facet. In the systematic approach to the analysis & description of the support technology facets of a domain one usually starts with a basically intrinsics facet-oriented domain description. We then suggest that the domain engineer “inquires” of every endurant and perishant whether it is an intrinsic entity or, perhaps a support technology.

### Railway Support Technology

**Example 90** We give a rough sketch description of possible rail unit switch technologies.

(i) In “ye olde” days, rail switches were “thrown” by manual labour, i.e., by railway staff assigned to and positioned at switches.

(ii) With the advent of reasonably reliable mechanics, pulleys and levers\(^4\) and steel wires, switches were made to change state by means of “throwing” levers in a cabin tower located centrally at the station (with the lever then connected through wires etc., to the actual switch).

(iii) This partial mechanical technology then emerged into electro-mechanics, and cabin tower staff was “reduced” to pushing buttons.

(iv) Today, groups of switches, either from a station arrival point to a station track, or from a station track to a station departure point, are set and reset by means also of electronics, by what is known as interlocking (for example, so that two different routes cannot be open in a station if they cross one another).

---

It must be stressed that Example 90 is just a rough sketch. In a proper narrative description the software (cum domain) engineer must describe, in detail, the subsystem of electronics, electro-mechanics and the human operator interface (buttons, lights, sounds, etc.). An aspect of supporting technology includes recording the state-behaviour in response to external stimuli. We give an example.

### Probabilistic Rail Switch Unit State Transitions

**Example 91** Figure 7.2 indicates a way of formalising this aspect of a supporting technology. Figure 7.2 intends to model the probabilistic (erroneous and correct) behaviour of a switch when subjected to settings (to switched (s) state) and re-settings (to direct (d) state). A switch may go to the switched state from the direct state when subjected to a switch setting s with probability psd.

![Diagram of Probabilistic Rail Switch Unit State Transitions](image)

**Input stimuli:**
- sw/psd
- w/1−psd−esd
- d/1−pdd−edd
- sw/ess
- sw/1−psd−ess
- sw/pss
- di/pdd
- di/eds
- di/edd
- di/1−pds−eds
- di/pds

**States:**
- s: Switched state
- d: Direct (reverted) state
- e: Error state

**Probabilities:**
- 0 < p < 1
- pss: Switching to switched state from switched state
- psd: Switching to switched state from direct state
- pds: Reverting to direct state from switched state
- esd: Switching to error state from direct state
- edd: Reverting to error state from direct state
- ess: Switching to error state from switched state
- eds: Reverting to error state from switched state

**Fig. 7.2. Probabilistic state switching**

### Traffic Signals

**Example 92** A traffic signal represents a technology in support of visualising hub states (transport net road intersection signaling states) and in effecting state changes.

276 A traffic signal, ts:TS, is here⁵ considered a part with observable hub states and hub state spaces. Hub states and hub state spaces are programmable, respectively static attributes of traffic signals.

277 A hub state space, hω, is a set of hub states such that each current hub state is in that hubs’ hub state space.

278 A hub state, hσ, is now modeled as a set of hub triples.

279 Each hub triple has a link identifier l₁ (“coming from”), a colour (red, yellow or green), and another link identifier l₂ (“going to”).

280 Signaling is now a sequence of one or more pairs of next hub states and time intervals, ti:TI, for example: \(<(hσ₁, t₁₁), (hσ₂, t₁₂), ... , (hσₙ₋₁, tᵢₙ₋₁), (hσₙ, tᵢₙ)>, n> > 0\).

The idea of a signaling is to first change the designated hub to state hσ₁, then wait ti₁ time units, then set the designated hub to state hσ₂, then wait ti₂ time units, et cetera, ending with final state σₙ and a
(supposedly) long time interval \( t_i \) before any decisions are to be made as to another signaling. The set of hub states \( \{ h_{\sigma_1}, h_{\sigma_2}, \ldots, h_{\sigma_{n-1}} \} \) of \( <(h_{\sigma_1}, t_{i_1}), (h_{\sigma_2}, t_{i_2}), \ldots, (h_{\sigma_{n-1}}, t_{i_{n-1}}), (h_{\sigma_n}, t_{i_n})> \), \( n > 0 \), is called the set of intermediate states. Their purpose is to secure an orderly phase out of green via yellow to red and phase in of red via yellow to green in some order for the various directions. We leave it to the reader to devise proper well-formedness conditions for signaling sequences as they depend on the hub topology.

281 A street signal (a semaphore) is now abstracted as a map from pairs of hub states to signaling sequences.

The idea is that given a hub one can observe its semaphore, and given the state, \( h_{\sigma} \) (not in the above set), of the hub “to be signaled” and the state \( h_{\sigma_{n}} \) into which that hub is to be signaled “one looks up” under that pair in the semaphore and obtains the desired signaling.

282 We treat hubs as processes with hub state spaces and semaphores as static attributes and hub states as programmable attributes. We ignore other attributes and input/outputs.

283 We can think of the change of hub states as taking place based the result of some internal, non-deterministic choice.
If a desired hub state change fails (i.e., does not meet the pre-condition, or for other reasons (e.g., failure of technology)), then we do not define the outcome of signaling.

We omit expression of a number of well-formedness conditions, e.g., that the htriple link identifiers are those of the corresponding mereology (lis), et cetera. The design of the semaphore, for a single hub or for a net of connected hubs has many similarities with the design of interlocking tables for railway tracks [230].

Another example shows another aspect of support technology: Namely that the technology must guarantee certain of its own behaviours, so that software designed to interface with this technology, together with the technology, meets dependability requirements.

**Railway Optical Gates**

**Example 93**  Train traffic (itt:iTF), intrinsically, is a total function over some time interval, from time (t:T) to continuously positioned (p:P) trains (tn:TN). Conventional optical gates sample, at regular intervals, the intrinsic train traffic. The result is a sampled traffic (stf:sTF). Hence the collection of all optical gates, for any given railway, is a partial function from intrinsic to sampled train traffics (stf). We need to express quality criteria that any optical gate technology should satisfy — relative to a necessary and sufficient description of a closeness predicate. The following axiom does that:

> For all intrinsic traffics, itf, and for all optical gate technologies, og, the following must hold: Let stt be the traffic sampled by the optical gates. For all time points, t, in the sampled traffic, those time points must also be in the intrinsic traffic, and, for all trains, tn, in the intrinsic traffic at that time, the train must be observed by the optical gates, and the actual position of the train and the sampled position must somehow be check-able to be close, or identical to one another.

Since units change state with time, n:N, the railway net, needs to be part of any model of traffic.
7.3.2 Requirements

Section 4.4 [Extension] of [66] illustrates a possible toll-gate, whose behaviour exemplifies a support technology. So do pumps of a pipe-line system such as illustrated in Examples 24, 29 and 42–44 in [78]. A pump of a pipe-line system gives rise to several forms of support technologies: from the Egyptian Shadoof [irrigation] pumps, and the Hellenic Archimedian screw pumps, via the 11th century Su Song pumps of China\(^6\), and the hydraulic “technologies” of Moorish Spain\(^7\) to the centrifugal and gear pumps of the early industrial age, et cetera, The techniques – to mention those that have influenced this author – of [380, 256, 301, 230] appears to apply well to the modeling of support technology requirements.

7.3.3 On Modeling Support Technologies

Support technologies in their relation to the domain in which they reside typically reflect real-time embeddedness. As such the techniques and languages for modeling support technologies resemble those for modeling event and process intensity, while temporal notions are brought into focus. Hence typical modeling notations include event-based languages (like Petri nets [332] or CSP) [238]), respectively process-based specification languages (like MSCs, [249], LSCs [200], Statecharts [199], or CSP) [238]), as well as temporal languages (like the Duration Calculus and [380] and Temporal Logic of Actions, TLA+) [263]).

7.4 Rules & Regulations

- **By a domain rule** we shall understand some text (in the domain) which prescribes how people or equipment are expected to behave when dispatching their duties, respectively when performing their functions.
- **By a domain regulation** we shall understand some text (in the domain) which prescribes what remedial actions are to be taken when it is decided that a rule has not been followed according to its intention.

The domain rules & regulations need or may not be explicitly present, i.e., written down. They may be part of the “folklore”, i.e., tacitly assumed and understood.

7.4.1 Conceptual Analysis

### Trains at Stations

**Example 94**

- Rule: In China the arrival and departure of trains at, respectively from, railway stations is subject to the following rule: 
  
  *In any three-minute interval at most one train may either arrive to or depart from a railway station.*

- Regulation: *If it is discovered that the above rule is not obeyed,* then there is some regulation which prescribes administrative or legal management and/or staff action, as well as some correction to the railway traffic.

### Trains Along Lines

**Example 95**

- Rule: In many countries railway lines (between stations) are segmented into blocks or sectors. The purpose is to stipulate that if two or more trains are moving along the line, then:


There must be at least one free sector (i.e., without a train) between any two trains along a line.

- Regulation: If it is discovered that the above rule is not obeyed, then there is some regulation which prescribes administrative or legal management and/or staff action, as well as some correction to the railway traffic.

At a meta-level, i.e., explaining the general framework for describing the syntax and semantics of the human-oriented domain languages for expressing rules and regulations, we can say the following: There are, abstractly speaking, usually three kinds of languages involved wrt. (i.e., when expressing) rules and regulations (respectively when invoking actions that are subject to rules and regulations). Two languages, Rules and Reg, exist for describing rules, respectively regulations; and one, Stimulus, exists for describing the form of the [always current] domain action stimuli. A syntactic stimulus, sy\_sti, denotes a function, se\_sti:STI: θ → θ, from any configuration to a next configuration, where configurations are those of the system being subjected to stimulations. A syntactic rule, sy\_rul:Rule, stands for, i.e., has as its semantics, its meaning, rul:RUL, a predicate over current and next configurations, (θ × θ) → Bool, where these next configurations have been brought about, i.e., caused, by the stimuli. These stimuli express: If the predicate holds then the stimulus will result in a valid next configuration.

A syntactic regulation, sy\_reg:Reg (related to a specific rule), stands for, i.e., has as its semantics, its meaning, a semantic regulation, se\_reg:REG, which is a pair. This pair consists of a predicate, pre\_reg:Pre\_REG, where Pre\_REG = (θ × θ) → Bool, and a domain configuration-changing function, act\_reg:Act\_REG, where Act\_REG = θ → θ, that is, both involving current and next domain configurations. The two kinds of functions express: If the predicate holds, then the action can be applied. The predicate is almost the inverse of the rules functions. The action function serves to undo the stimulus function.

The idea is now the following: Any action (i.e., event) of the system, i.e., the application of any stimulus, may be an action (i.e., event) in accordance with the rules, or it may not. Rules therefore express whether stimuli are valid or not in the current configuration. And regulations therefore express whether they should be applied, and, if so, with what effort. More specifically, there is usually, in any current system configuration, given a set of pairs of rules and regulations. Let (sy\_rul,sy\_reg) be any such pair. Let sy\_sti be any possible stimulus. And let θ be the current configuration. Let the stimulus, sy\_sti, applied in that configuration result in a next configuration, θ′, where θ′ = (meaning(sy\_sti))(θ). Let θ′ violate the rule, ¬valid(sy\_sti,sy\_rul)(θ), then if predicate part, pre\_reg, of the meaning of the regulation, sy\_reg, holds in that violating next configuration, pre\_reg(θ′,(meaning(sy\_sti))(θ)), then the action part, act\_reg, of the meaning of the regulation, sy\_reg, must be applied, act\_reg(θ), to remedy the situation.
axiom
\[ \forall (sy_{rul}, sy_{reg}) \colon \text{Rule} \land \text{Reg} \cdot \\
\text{let } se_{rul} = \text{meaning}(sy_{rul}), \\
(\text{pre}_{reg}, \text{act}_{reg}) = \text{meaning}(sy_{reg}) \text{ in} \\
\forall sy_{sti} : \text{Stimulus}, \, \theta : \Theta \cdot \\
\sim \text{valid}(sy_{sti}, se_{rul})(\theta) \Rightarrow \\
\sim \text{pre}_{reg}(\theta,(\text{meaning}(sy_{sti}))(\theta)) \Rightarrow \\
\exists n \theta : \Theta \cdot \text{act}_{reg}(\theta) = n \theta \land se_{rul}(\theta, n \theta) \]
end

It may be that the regulation predicate fails to detect applicability of regulations actions. That is, the interpretation of a rule differs, in that respect, from the interpretation of a regulation. Such is life in the domain, i.e., in actual reality.

7.4.2 Requirements

Implementation of rules & regulations implies monitoring and partially controlling the states symbolised by \( \Theta \) in Sect. 7.4.1. Thus some partial implementation of \( \Theta \) must be required; as must some monitoring of states \( \theta : \Theta \) and implementation of the predicates meaning, valid, interpret, pre_reg and action(s) act_reg. The emerging requirements follow very much in the line of support technology requirements.

7.4.3 On Modeling Rules and Regulations

Usually rules (as well as regulations) are expressed in terms of domain entities, including those grouped into “the state”, functions, events, and behaviours. Thus the full spectrum of model-ling techniques and notations may be needed. Since rules usually express properties one often uses some combination of axioms and wellformedness predicates. Properties sometimes include temporality and hence temporal notations (like Duration Calculus or Temporal Logic of Actions) are used. And since regulations usually express state (restoration) changes one often uses state changing notations (such as found in Allard [251], B or event-B [1], RSL [176], VDM-SL [88, 89, 154], and Z [374]). In some cases it may be relevant to model using some constraint satisfaction notation [9] or some Fuzzy Logic notations [354].

7.5 Scripts

• By a domain script we shall understand the structured, almost, if not outright, formally expressed, wording of a procedure on how to proceed, one that has legally binding power, that is, which may be contested in a court of law.

7.5.1 Conceptual Analysis

Rules & regulations are usually expressed, even when informally so, as predicates. Scripts, in their procedural form, are like instructions, as for an algorithm.

### Example 96

Our formulation amounts to just a (casual) rough sketch. It is followed by a series of four large examples. Each of these elaborate on the theme of (bank) scripts. The problem area is that of how repayments of mortgage loans are to be calculated. At any one time a mortgage loan has a balance, a most recent previous date of repayment, an interest rate and a handling fee. When a repayment occurs, then the following calculations shall take place: (i) the interest on the balance of the loan since the most recent repayment, (ii) the handling fee, normally considered fixed, (iii) the effective repayment — being the difference between the repayment and the sum of the interest and the handling fee — and the new balance, being the difference between the old balance and the effective repayment. We assume repay-
ments to occur from a designated account, say a demand/deposit account. We assume that bank to have designated fee and interest income accounts. (i) The interest is subtracted from the mortgage holder’s demand/deposit account and added to the bank’s interest (income) account. (ii) The handling fee is subtracted from the mortgage holder’s demand/deposit account and added to the bank’s fee (income) account. (iii) The effective repayment is subtracted from the mortgage holder’s demand/deposit account and also from the mortgage balance. Finally, one must also describe deviations such as overdue repayments, too large, or too small repayments, and so on.

A Formally Described Bank Script

Example 97 First we must informally and formally define the bank state: There are clients (c:C), account numbers (a:A), mortgage numbers (m:M), account yields (ay:A Y) and mortgage interest rates (mi:MI). The bank registers, by client, all accounts (ρ:A_Register) and all mortgages (µ:M_Register). To each account number there is a balance (α:Accounts). To each mortgage number there is a loan (ℓ:Loans). To each loan is attached the last date that interest was paid on the loan.

value
r, r’:Real axiom ...

value
ay:AY, mi:MI

We — perhaps too rigidly — assume that mortgage interest rates are higher than demand/deposit account interest rates: ay<mi. Operations on banks are denoted by the commands of the bank script language. First the syntax:

type
Cmd = OpA | CloA | Dep | Wdr | OpM | CloM | Pay
OpA == mkOA(c:C)
CloA == mkCA(c:C,a:A)
Dep == mkD(c:C,a:A,p:P)
Wdr == mkW(c:C,a:A,p:P)
OpM == mkOM(c:C,m:M)
Pay == mkPM(c:C,a:A,m:M,p:P,d:Date)
CloM == mkCM(c:C,m:M,p:P)
Reply == A | M | P | OkNok
OkNok == ok | notok

value
  period: Date × Date → Days [for calculating interest]
  before: Date × Date → Bool [first date is earlier than last date]

And then the semantics:

$$\text{int}_\text{Cmd}(\text{mkPM}(c,a,m,p,d))((\rho,\alpha,\mu,\ell) \equiv$$

$$\begin{align*}
&\text{let } (b,d') = \ell(m) \text{ in} \\
&\text{if } \alpha(a) \geq p \\
&\text{then} \\
&\text{let } i = \text{interest}(m,b,\text{period}(d,d')), \\
&\ell' = \ell \uplus [m \mapsto \ell(m) - (p - i)] \\
&\alpha' = \alpha \uplus [a \mapsto \alpha(a) - p, a_i \mapsto \alpha(a_i) + i] \text{ in} \\
&((\rho,\alpha',\mu,\ell'),\text{ok}) \text{ end} \\
&\text{else} \\
&((\rho,\alpha',\mu,\ell'),\text{nok}) \text{ end end}
\end{align*}$$

pre $c \in \text{dom } \mu \land a \in \text{dom } \alpha \land m \in \mu(c)$

post before(d,d')

interest: MI × Loan × Days → P

The idea about scripts is that they can somehow be objectively enforced: that they can be precisely understood and consistently carried out by all stakeholders, eventually leading to computerisation. But they are, at all times, part of the domain.

### 7.5.2 Requirements

Script requirements call for the possibly interactive computerisation of algorithms, that is, for rather classical computing problems. But sometimes these scripts can be expressed, computably, in the form of programs in a domain specific language. As an example we refer to [124]. [124] illustrates how the design of pension and life insurance products, and their administration, reserve calculations, and audit, can be based on a common formal notation. The notation is human-readable and machine-processable, and specialised to the actuarial domain, achieving great expressive power combined with ease of use and safety. More specifically (a) product definitions based on standard actuarial models, including arbitrary continuous-time Markov and semi-Markov models, with cyclic transitions permitted; (b) calculation descriptions for reserves and other quantities of interest, based on differential equations; and (c) administration rules.

### 7.5.3 On Modeling Scripts

Scripts (as are licenses) are like programs (respectively like prescriptions program executions). Hence the full variety of techniques and notations for modeling programming (or specification) languages apply [14, 187, 334, 339, 352, 367]. [40, Chaps. 6–9] cover pragmatics, semantics and syntax techniques for defining functional, imperative and concurrent programming languages.

### 7.6 License Languages

**License**: a right or permission granted in accordance with law by a competent authority to engage in some business or occupation,
to do some act, or to engage in some transaction which but for such license would be unlawful.

Merriam Webster Online [286]

7.6.1 Conceptual Analysis

7.6.1.1 The Settings

A special form of scripts are increasingly appearing in some domains, notably the domain of electronic, or digital media. Here licenses express that a licensor, o, permits a licensee, u, to render (i.e., play) works of proprietary nature CD ROM-like music, DVD-like movies, etc. while obligating the licensee to pay the licensor on behalf of the owners of these, usually artistic works. Classical digital rights license languages, [27, 11, 121, 122, 123, 246, 117, 186, 196, 270, 293, 289, 272, 262, 337, 326, 325, 4, 294], applied to the electronic “downloading”, payment and rendering (playing) of artistic works (for example music, literature readings and movies). In this chapter we generalise such applications languages and we extend the concept of licensing to also cover work authorisation (work commitment and promises) in health care, public government and schedule transport. The digital works for these new application domains are patient medical records, public government documents and bus/train/aircraft transport contracts. Digital rights licensing for artistic works seeks to safeguard against piracy and to ensure proper payments for the rights to render these works. Health care and public government license languages seek to ensure transparent and professional (accurate and timely) health care, respectively ‘good governance’. Transport contract languages seeks to ensure timely and reliable transport services by an evolving set of transport companies. Proper mathematical definition of licensing languages seeks to ensure smooth and correct computerised management of licenses and contracts.

7.6.1.2 On Licenses

The concepts of licenses and licensing express relations between (i) actors (licensors (the authority) and licensees), (ii) entities (artistic works, hospital patients, public administration, citizen documents) and bus transport contracts and (iii) functions (on entities), and as performed by actors. By issuing a license to a licensee, a licensor wishes to express and enforce certain permissions and obligations: which functions on which entities the licensee is allowed (is licensed, is permitted) to perform. In this chapter we shall consider four kinds of entities: (i) digital recordings of artistic and intellectual nature: music, movies, readings (“audio books”), and the like, (ii) patients in a hospital as represented also by their patient medical records, (iii) documents related to public government, and (iv) transport vehicles, time tables and transport nets (of a buses, trains and aircraft).

7.6.1.3 Permissions and Obligations

The permissions and obligations issues are, (1) for the owner (agent) of some intellectual property to be paid (an obligation) by users when they perform permitted operations (rendering, copying, editing, sub-licensing) on their works; (2) for the patient to be professionally treated — by medical staff who are basically obliged to try to cure the patient; (3) for public administrators and citizens to enjoy good governance: transparency in law making (national parliaments and local prefectures and city councils), in law enforcement (i.e., the daily administration of laws), and law interpretation (the judiciary) — by agents who are basically obliged to produce certain documents while being permitted to consult (i.e., read, perhaps copy) other documents; and (4) for bus passengers to enjoy reliable bus schedules — offered by bus transport companies on contract to, say public transport authorities and on sub-contract to other such bus transport companies where these transport companies are obliged to honour a contracted schedule.
7.6.2 The Pragmatics

By pragmatics we understand the study and practice of the factors that govern our choice of language in social interaction and the effects of our choice on others.

In this section we shall rough-sketch-describe pragmatic aspects of the four domains of (1) production, distribution and consumption of artistic works, (2) the hospitalisation of patient, i.e., hospital health care, (3) the handling of law-based document in public government and (4) the operational management of schedule transport vehicles. The emphasis is on the pragmatics of the terms, i.e., the language used in these four domains.

7.6.2.1 Digital Media

Example 98 The intrinsic entities of the performing arts are the artistic works: drama or opera performances, music performances, readings of poems, short stories, novels, or jokes, movies, documentaries, newsreels, etc. We shall limit our span to the scope of electronic renditions of these artistic works: videos, CDs or other. In this chapter we shall not touch upon the technical issues of “downloading” (whether “streaming” or copying, or other). That and other issues should be analysed in [375].

7.6.2.1.1 Operations on Digital Works:

For a consumer to be able to enjoy these works that consumer must (normally first) usually “buy a ticket” to their performances. The consumer, i.e., the theatre, opera, concert, etc., “goer” (usually) cannot copy the performance (e.g., “tape it”), let alone edit such copies of performances. In the context of electronic, i.e., digital renditions of these performances the above “cannots” take on a new meaning. The consumer may copy digital recordings, may edit these, and may further pass on such copies or editions to others. To do so, while protecting the rights of the producers (owners, performers), the consumer requests permission to have the digital works transferred (“downloaded”) from the owner/producer to the consumer, so that the consumer can render (“play”) these works on own rendering devices (CD, DVD, etc., players), possibly can copy all or parts of them, then possibly can edit all or parts of the copies, and, finally, possibly can further license these “edited” versions to other consumers subject to payments to “original” licensor.

7.6.2.1.2 License Agreement and Obligation:

To be able to obtain these permissions the user agrees with the wording of some license and pays for the rights to operate on the digital works.

7.6.2.1.3 Two Assumptions:

Two, related assumptions underlie the pragmatics of the electronics of the artistic works. The first assumption is that the format, the electronic representation of the artistic works is proprietary, that is, that the producer still owns that format. Either the format is publicly known or it is not, that is, it is somehow “secret”. In either case we “derive” the second assumption (from the fulfillment of the first). The second assumption is that the consumer is not allowed to, or cannot operate on the works by own means (software, machines). The second assumption implies that acceptance of a license results in the consumer
receiving software that supports the consumer in performing all operations on licensed works, their copies and edited versions: rendering, copying, editing and sub-licensing.

### 7.6.2.1.4 Protection of the Artistic Electronic Works:

The issue now is: how to protect the intellectual property (i.e., artistic) and financial (exploitation) rights of the owners of the possibly rendered, copied and edited works, both when, and when not further distributed.

### 7.6.2.2 Health-care

**Example 99** Citizens go to hospitals in order to be treated for some calamity (disease or other), and by doing so these citizens become patients. At hospitals patients, in a sense, issue a request to be treated with the aim of full or partial restitution. This request is directed at medical staff, that is, the patient authorises medical staff to perform a set of actions upon the patient. One could claim, as we shall, that the patient issues a license.

#### 7.6.2.2.1 Patients and Patient Medical Records:

So patients and their attendant patient medical records (PMRs) are the main entities, the “works” of this domain. We shall treat them synonymously: PMRs as surrogates for patients. Typical actions on patients — and hence on PMRs — involve admitting patients, interviewing patients, analysing patients, diagnosing patients, planning treatment for patients, actually treating patients, and, under normal circumstance, to finally release patients.

#### 7.6.2.2.2 Medical Staff:

Medical staff may request (‘refer’ to) other medical staff to perform some of these actions. One can conceive of describing action sequences (and ‘referrals’) in the form of hospitalisation (not treatment) plans. We shall call such scripts for licenses.

#### 7.6.2.2.3 Professional Health Care:

The issue is now, given that we record these licenses, their being issued and being honoured, whether the handling of patients at hospitals follow, or does not follow properly issued licenses.

### 7.6.2.3 Government Documents

**Example 100** By public government we shall, following Charles de Secondat, baron de Montesquieu (1689–1755)⁹, understand a composition of three powers: the law-making (legislative), the law-enforcing and the law-interpreting parts of public government. Typically national parliament and local (province and city) councils are part of law-making government. Law-enforcing government is called the executive (the administration). And law-interpreting government is called the judiciary [system] (including lawyers etc.).

#### 7.6.2.3.1 Documents:

A crucial means of expressing public administration is through documents.⁰ We shall therefore provide a brief domain analysis of a concept of documents. (This document domain description also applies to
patient medical records and, by some “light” interpretation, also to artistic works — insofar as they also are documents.) Documents are created, edited and read; and documents can be copied, distributed, the subject of calculations (interpretations) and be shared and shredded.

7.6.2.3.2 Document Attributes:

With documents one can associate, as attributes of documents, the actors who created, edited, read, copied, distributed, shared, performed calculations and shredded documents. With these operations on documents, and hence as attributes of documents one can, again conceptually, associate the location and time of these operations.

7.6.2.3.3 Actor Attributes and Licenses:

With actors (whether agents of public government or citizens) one can associate the authority (i.e., the rights) these actors have with respect to performing actions on documents. We now intend to express these authorisations as licenses.

7.6.2.3.4 Document Tracing:

An issue of public government is whether citizens and agents of public government act in accordance with the laws — with actions and laws reflected in documents such that the action documents enables a trace from the actions to the laws “governing” these actions. We shall therefore assume that every document can be traced back to its law-origin as well as to all the documents any one document-creation or -editing was based on.

7.6.2.4 Transportation

Transportation is one of the prime areas for domain analysis & description: roads and vehicles: private automobiles, buses, trucks, etc., aircraft, shipping, trains.

Example 101

7.6.2.4.1 A Synopsis:

Contracts obligate transport companies to deliver bus traffic according to a timetable. The timetable is part of the contract. A contractor may sub-contract (other) transport companies to deliver bus traffic according to timetables that are sub-parts of their own timetable. Contractors are either public transport authorities or contracted transport companies. Contracted transport companies may cancel a subset of bus rides provided the total amount of cancellations per 24 hours for each bus line does not exceed a contracted upper limit The cancellation rights are spelled out in the contract. A sub-contractor cannot increase a contracted upper limit for cancellations above what the sub-contractor was told (in its contract) by its contractor. Et cetera.

7.6.2.4.2 A Pragmatics and Semantics Analysis:

The “works” of the bus transport contracts are two: the timetables and, implicitly, the designated (and obligated) bus traffic. A bus timetable appears to define one or more bus lines, with each bus line giving rise to one or more bus rides. Nothing is (otherwise) said about regularity of bus rides. It appears that bus ride cancellations must be reported back to the contractor. And we assume that cancellations by a sub-contractor is further reported back also to the sub-contractor’s contractor. Hence eventually that the public transport authority is notified. Nothing is said, in the contracts, such as we shall model them, about passenger fees for bus rides nor of percentages of profits (i.e., royalties) to be paid back from a sub-contractor to the contractor. So we shall not bother, in this example, about transport costs nor transport.
subsides. But will leave that necessary aspect as an exercise. The opposite of cancellations appears to be ‘insertion’ of extra bus rides, that is, bus rides not listed in the time table, but, perhaps, mandated by special events. We assume that such insertions must also be reported back to the contractor. We assume concepts of acceptable and unacceptable bus ride delays. Details of delay acceptability may be given in contracts, but we ignore further descriptions of delay acceptability. but assume that unacceptable bus ride delays are also to be (iteratively) reported back to contractors. We finally assume that sub-contractors cannot (otherwise) change timetables. (A timetable change can only occur after, or at, the expiration of a license.) Thus we find that contracts have definite period of validity. (Expired contracts may be replaced by new contracts, possibly with new timetables.)

7.6.3.2.2 Revoke Licenses:

Example 102

7.6.3.2.1 Digital Media:

For digital media the original licensors are the original producers of music, film, etc. The “original” licensees are you and me! Thereafter some of us may become licensors, etc.

7.6.3.2.2 Health-care:

For health-care the original licensors are, say in Denmark, the Danish governments’ National Board of Health; and the “original” licensees are the national hospitals. These then sub-license their medical clin-
ics (rheumatology, cancer, urology, gynecology, orthopedics, neurology, etc.) which again sub-licenses their medical staff (doctors, nurses, etc.). A medical doctor may, as is the case in Denmark for certain actions, not [necessarily] perform these but may sub-license their execution to nurses, etc.

7.6.3.2.3 Documents:

For government documents the original licensor are the (i) heads of parliament, regional and local governments, (ii) government (prime minister) and the heads of respective ministries, respectively the regional and local agencies and administrations. The “original” licensees are (i′) the members of parliament, regional and local councils charged with drafting laws, rules and regulations, (ii′) the ministry, respectively the regional and local agency department heads. These (the ′s) then become licensors when licensing their staff to handle specific documents.

7.6.3.2.4 Transport:

For scheduled passenger (etc.) transportation the original licensors are the state, regional and/or local transport authorities. The “original” licensees are the public and private transport firms. These latter then become licensors licensing drivers to handle specific transport lines and/or vehicles.

7.6.3.3 Actors and Actions

In preparation for Example 103 we show Figure 7.3.

![Diagram of Actors and Actions](image)

Fig. 7.3. An example single-illness non-fatal hospitalisation plan. States: \{1,2,3,4,5,6,7,8,9\}
7.6.3.3.1 Digital Media:

\( w \) refers to a digital “work” with \( w' \) designating a newly created one; \( s_i \) refers to a sector of some work.

- **render** \( w(s_i, s_j, \ldots, s_k) \):
  - \( s_i, s_j, \ldots, s_k \) of work \( w \)
  - are rendered (played, visualised) in that order.

- **copy** \( w(s_i, s_j, \ldots, s_k) \):
  - \( s_i, s_j, \ldots, s_k \) of work \( w \)
  - are copied and becomes work \( w' \).

- **edit** \( w \) with \( E(w_\alpha(s_a, s_b, \ldots, s_c), \ldots, w_\gamma(s_p, s_q, \ldots, s_r)) \):
  - work \( w \) is edited while [also] incorporating references to or excerpts from [other] works
  - \( w_\alpha(s_a, s_b, \ldots, s_c), \ldots, w_\gamma(s_p, s_q, \ldots, s_r) \).

- **read** \( w \):
  - work \( w \) is read, i.e., information about work \( w \) is somehow displayed.

- **\( \ell \) : licensor \( m \) contracts licensees \( \{u_1, u_2, \ldots, u_u\} \)
  - to perform actions \( \{\text{RENDER, COPY, EDIT, READ}\} \)
  - on work items \( \{w_{i_1}, w_{i_2}, \ldots, w_{i_w}\} \).

Et cetera: other forms of actions can be thought of.

7.6.3.3.2 Health-care:

- Actors are here limited to the patients and the medical staff.
- We refer to Fig. 7.3 on the previous page.
- It shows an archetypal hospitalisation plan.
  - \( \pi \) designates patients,
  - \( t \) designates treatment (medication, surgery, . . . ).
- Actions are performed by medical staff, say \( h \), with \( h \) being an implicit argument of the actions.

- **interview** \( \pi \): a PMR with name, age, family relations, addresses, etc., is established for patient \( \pi \).
- **admit** \( \pi \): the PMR records the anamnese (medical history) for patient \( \pi \).
- **establish analysis plan** \( \pi \): the PMR records which analyses (blood tests, ECG, blood pressure, etc.) are to be carried out.
- **analyse** \( \pi \): the PMR records the results of the analyses referred to previously.
- **diagnose** \( \pi \): medical staff \( h \) diagnoses, based on the analyses most recently performed.
- **plan treatment for** \( \pi \): medical staff \( h \) sets up a treatment plan for patient \( \pi \) based on the diagnosis most recently performed.
- **treat** \( \pi \) wrt. \( t \): medical staff \( h \) performs treatment \( t \) on patient \( \pi \), observes “reaction” and records this in the PMR. Predicate “actions”:
  - **more analysis** \( \pi \)?.
  - **more treatment** \( \pi \)? and
  - **more diagnosis** \( \pi \)?.
- **release** \( \pi \): either the patient dies or is declared ready to be sent ’home’.
- **\( \ell \) : licensor \( o \) contracts medical staff \( \{m_{m_1}, m_{m_2}, \ldots, m_{m_m}\} \)
  - to perform actions
    - \( \{\text{INTERVIEW, PLAN ANALYSIS, DIAGNOSE, TREAT, ADMIT, ANALYSE, PLAN TREATMENT, RELEASE}\} \)
  - on patients \( \{\pi_{p_1}, \pi_{p_2}, \ldots, \pi_{p_p}\} \).
Et cetera: other forms of actions can be thought of.

7.6.3.3.3 Documents:

$d$ refer to documents with $d'$ designating new documents.

- $d' \equiv \text{create based on } d_1, d_2, \ldots, d_n$: A new document, named $d'$, is created, with no information “contents”, but referring to existing documents $d_1, d_2, \ldots, d_n$.
- edit $d$ with $\mathcal{E}$ based on $d_{a_1}, d_{a_2}, \ldots, d_{a_n}$: document $d$ is edited with $\mathcal{E}$ being the editing function and $\mathcal{E}^{-1}$ being its “undo” inverse.
- read $d$: document $d$ is being read.
- shred $d$: document $d$ is shredded. That is, no more actions can be performed on $d$.
- $\ell : \text{licensor} \circ \text{contracts} \text{civil service staff} \{e_1, e_2, \ldots, e_n\} \text{to perform actions} \{\text{CREATE, EDIT, READ, COPY, FREEZE, SHRED}\} \text{on documents} \{d_1, d_2, \ldots, d_m\}$.

Et cetera: other forms of actions can be thought of.

7.6.3.4 Transport:

- We restrict, without loss of generality, to bus transport.
  - There is a timetable, $tt$.
  - It records bus lines $l$, and specific instances of bus rides, $b$.
- These are some archetypal operations:
  - start bus ride $l, b$ at time $t$: Bus line $l$ is recorded in $tt$ and its departure in $tt$ is recorded as $\tau$. Starting that bus ride at $t$ means that the start is either on time, i.e., $t=\tau$, or the start is delayed $\delta_l : \tau - t$ or advanced $\delta_l : t - \tau$ where $\delta_l$ and $\delta_b$ are expected to be small intervals. All this is to be reported, in due time, to the contractor.
  - end bus ride $l, b$ at time $t$: Ending bus ride $l, b$ at time $t$ means that it is either ended on time, or earlier, or delayed. This is to be reported, in due time, to the contractor.
  - cancel bus ride $l, b$ at time $t$: $t$ must be earlier than the scheduled departure of bus ride $l, b$.
  - insert an extra bus $l, b'$ at time $t$: $t$ must be the same time as the scheduled departure of bus ride $l, b$ with $b'$ being a “marked” version of $b$.
- $\ell : \text{licensor} \circ \text{contracts} \text{transport staff} \{b_{e_1}, b_{e_2}, \ldots, b_{e_n}\} \text{to perform actions} \{\text{START, END, CANCEL, INSERT}\} \text{on work items} \{e_1, e_2, \ldots, e_n\}$.

Et cetera: other forms of actions can be thought of.

7.6.4 Requirements

Requirements for license language implementation basically amounts to requirements for three aspects. (i) The design of the license language, its abstract and concrete syntax, its interpreter, and its interfaces to distributed licensor and licensee behaviours; (ii) the requirements for a distributed system of licensor and licensee behaviours; and (iii) the monitoring and partial control of the states of licensor and licensee behaviours. The structuring of these distributed licensor and licensee behaviours differ from slightly to somewhat, but not that significant in the four license languages examples. Basically the licensor and licensee behaviours form a set of behaviours. Basically everyone can communicate with everyone. For the case of digital media licensee behaviours communicate back to licensor behaviours whenever a properly licensed action is performed — resulting in the transfer of funds from licensees to licensors. For the case of health care some central authority is expected to validate the granting of licenses and appear to be bound by medical training. For the case of documents such checks appear to be bound by predetermined authorisation rules. For the case of transport one can perhaps speak of more rigid management & organisation dependencies as licenses are traditionally transferred between independent authorities and companies.
7.6.5 On Modeling License Languages

Licensors are expected to maintain a state which records all the licenses it has issued. Whenever at licensee “reports back” (the begin and/or the end) of the performance of a granted action, this is recorded in its state. Sometimes these granted actions are subject to fees. The licensor therefore calculates outstanding fees — etc. Licensees are expected to maintain a state which records all the licenses it has accepted. Whenever an action is to be performed the licensee records this and checks that it is permitted to perform this action. In many cases the licensee is expected to “report back”, both the beginning and the end of performance of that action, to the licensor. A typical technique of modeling licensors, licensees and patients, i.e., their PMRs, is to model them as (never ending) processes, a la CSP [238], with input/output, $\text{ch} ?/ \text{ch} ! m$, communications between licensors, licensees and PMRs. Their states are modeled as programmable attributes.

7.7 Management & Organisation

• By domain management we shall understand such people (such decisions) (i) who (which) determine, formulate and thus set standards (cf. rules and regulations, Sect. 7.4) concerning strategic, tactical and operational decisions; (ii) who ensure that these decisions are passed on to (lower) levels of management and to floor staff; (iii) who make sure that such orders, as they were, are indeed carried out; (iv) who handle undesirable deviations in the carrying out of these orders cum decisions; and (v) who “backstops” complaints from lower management levels and from “floor” staff.

• By domain organisation we shall understand (vi) the structuring of management and non-management staff “oversee-able” into clusters with “tight” and “meaningful” relations; (vii) the allocation of strategic, tactical and operational concerns to within management and non-management staff clusters; and hence (viii) the “lines of command”: who does what, and who reports to whom, administratively and functionally.

The ‘&’ is justified from the interrelations of items (i–viii).

7.7.1 Conceptual Analysis

We first bring some examples.

**Train Monitoring, I**

**Example 104** In China, as an example, till the early 1990s, rescheduling of trains occurs at stations and involves telephone negotiations with neighbouring stations (“up and down the lines”). Such rescheduling negotiations, by phone, imply reasonably strict management and organisation (M&O). This kind of M&O reflects the geographical layout of the rail net.

**Railway Management and Organisation: Train Monitoring, II**

**Example 105** We single out a rather special case of railway management and organisation. Certain (lowest-level operational and station-located) supervisors are responsible for the day-to-day timely progress of trains within a station and along its incoming and outgoing lines, and according to given timetables. These supervisors and their immediate (middle-level) managers (see below for regional managers) set guidelines (for local station and incoming and outgoing lines) for the monitoring of train traffic, and for controlling trains that are either ahead of or behind their schedules. By an incoming and an outgoing line we mean part of a line between two stations, the remaining part being handled by neighbouring station management. Once it has been decided, by such a manager, that a train is not following its schedule, based on information monitored by non-management staff, then that manager directs that staff: (i) to suggest a new schedule for the train in question, as well as for possibly affected other trains, (ii) to negotiate the new schedule with appropriate neighbouring stations, until a proper reschedule can be de-
cided upon, by the managers at respective stations, (iii) and to enact that new schedule. A (middle-level operations) manager for regional traffic, i.e., train traffic involving several stations and lines, resolves possible disputes and conflicts.

The above, albeit rough-sketch description, illustrated the following management and organisation issues: (i) There is a set of lowest-level (as here: train traffic scheduling and rescheduling) supervisors and their staff; (ii) they are organised into one such group (as here: per station); (iii) there is a middle-level (as here: regional train traffic scheduling and rescheduling) manager (possibly with some small staff), organised with one such per suitable (as here: railway) region; and (iv) the guidelines issued jointly by local and regional (...) supervisors and managers imply an organisational structuring of lines of information provision and command.

People staff enterprises, the components of infrastructures with which we are concerned, i.e., for which we develop software. The larger these enterprises — these infrastructure components — the more need there is for management and organisation. The role of management is roughly, for our purposes, twofold: first, to perform strategic, tactical and operational work, to set strategic, tactical and operational policies — and to see to it that they are followed. The role of management is, second, to react to adverse conditions, that is, to unforeseen situations, and to decide how they should be handled, i.e., conflict resolution. Policy setting should help non-management staff operate normal situations — those for which no management interference is thus needed. And management “backstops” problems: management takes these problems off the shoulders of non-management staff. To help management and staff know who’s in charge wrt. policy setting and problem handling, a clear conception of the overall organisation is needed. Organisation defines lines of communication within management and staff, and between these. Whenever management and staff has to turn to others for assistance they usually, in a reasonably well-functioning enterprise, follow the command line: the paths of organigrams — the usually hierarchical box and arrow/line diagrams.

The management and organisation model of a domain is a partial specification; hence all the usual abstraction and modeling principles, techniques and tools apply. More specifically, management is a set of predicate functions, or of observer and generator functions. These either parametrise other, the operations functions, that is, determine their behaviour, or yield results that become arguments to these other functions. Organisation is thus a set of constraints on communication behaviours. Hierarchical, rather than linear, and matrix structured organisations can also be modeled as sets (of recursively invoked sets) of equations.

To relate classical organigrams to formal descriptions we first show such an organigram (Fig. 7.4 on the following page), and then we show schematic processes which — for a rather simple scenario — model managers and the managed!

Based on such a diagram, and modeling only one neighbouring group of a manager and the staff working for that manager we get a system in which one manager, \( \text{mgr} \), and many staff, \( \text{stf} \), coexist or work concurrently, i.e., in parallel. The \( \text{mgr} \) operates in a context and a state modeled by \( \psi \). Each staff, \( \text{stf}(i) \) operates in a context and a state modeled by \( \sigma(i) \).
new state. The manager chooses — in this model — which of the two things (1 or 2) to do by a so-called non-deterministic internal choice ($\lceil\rceil$).

$$m: \Psi \rightarrow \text{in, out } \{\text{ms}[i]|i:Sx\} \text{ Unit}$$

$$\text{mgr}(\psi) \equiv$$

(1) let $(\psi',m) = m_{\text{out}}(\psi) \text{ in } \{\text{ms}[i]|i:Sx\}; \text{mgr}(\psi') \text{ end}$

(2) let $\psi' = \lceil\rceil \{m=\text{ms}[i]\? \text{ in } m_{\text{in}}(i,m)(\psi) \text{ end }|i:Sx\} \text{ in } \text{mgr}(\psi') \text{ end}$

$m_{\text{out}}: \Psi \rightarrow \Psi \times \text{MSG},$

$m_{\text{in}}: Sx \times \text{MSG} \rightarrow \Psi \rightarrow \Psi$

And in this system, staff $i$, $\text{stf}(i)$, (1) either is willing to receive a message, $\text{msg}$, from the manager, and then to change, $\text{st}_{\text{in}}(\text{msg})(\sigma)$, state accordingly, or (2) to concoct, $\text{st}_{\text{out}}(\sigma)$, a message, $\text{msg}$ (thus changing state) for the manager, and send it $\text{ms}[i]|\text{msg}$. In both cases the staff resumes work as from the new state. The staff member chooses — in this model — which of the two “things” (1 or 2) to do by a non-deterministic internal choice ($\lceil\rceil$).

$$\text{stf}: i:Sx \rightarrow \Sigma \rightarrow \text{in, out } \text{ms}[i] \text{ Unit}$$

$$\text{stf}(i)(\sigma) \equiv$$

(1) let $m=\text{ms}[i]? \text{ in } \text{stf}(i)(\text{stf}_{\text{in}}(m)(\sigma)) \text{ end}$

(2) let $(\sigma',m) = \text{st}_{\text{out}}(\sigma) \text{ in } \text{ms}[i]|m; \text{stf}(i)(\sigma') \text{ end}$

$$\text{st}_{\text{in}}: \text{MSG} \rightarrow \Sigma \rightarrow \Sigma,$$

$$\text{st}_{\text{out}}: \Sigma \rightarrow \Sigma \times \text{MSG}$$

Both manager and staff processes recurse (i.e., iterate) over possibly changing states. The management process non-deterministically, internal choice, “alternates” between “broadcast”-issuing orders to staff and receiving individual messages from staff. Staff processes likewise non-deterministically, internal choice, alternate between receiving orders from management and issuing individual messages to management. The conceptual example also illustrates modeling stakeholder behaviours as interacting (here CSP-like) processes.
Example 106 We think of (i) strategic, (ii) tactic, and (iii) operational managers as well as (iv) supervisors, (v) team leaders and the rest of the (vi) staff (i.e., workers) of a domain enterprise as functions. Each category of staff, i.e., each function, works in state and updates that state according to schedules and resource allocations — which are considered part of the state. To make the description simple we do not detail the state other than saying that each category works on an “instantaneous copy” of “the” state. Now think of six staff category activities, strategic managers, tactical managers, operational managers, supervisors, team leaders and workers as six simultaneous sets of actions. Each function defines a step of collective (i.e., group) (strategic, tactical, operational) management, supervisor, team leader and worker work. Each step is considered “atomic.” Now think of an enterprise as the “repeated” step-wise simultaneous performance of these category activities. Six “next” states arise. These are, in the reality of the domain, ameliorated, that is reconciled into one state. However with the next iteration, i.e., step, of work having each category apply its work to a reconciled version of the state resulting from that category’s previously yielded state and the mediated “global” state. Caveat: The below is not a mathematically proper definition. It suggests one!

```plaintext
type
  0. Σ, Σt, Σu, Σw, Σo, Σe
value
  1. str, tac, opr, sup, tea, wrk: Σi → Σi
  2. str, tac, opr, sup, tea, wrk: Σ → (Σt1 × Σu1 × Σv1 × Σw1) → Σ
  3. objective: (Σt × Σu × Σv × Σw × Σo) → Bool
  4. enterprise, ameliorate: (Σt × Σu × Σv × Σw × Σo) → Σ
  5. let σ′ = stra(str(σ1))(σ′t, σ′u, σ′w, σ′o, σ′e).
  6. σ′t = tac(σ′t)(σ′t, σ′u, σ′w, σ′o, σ′e).
  7. σ′u = opr(σ′u)(σ′t, σ′u, σ′w, σ′o, σ′e).
  8. σ′w = supr(σ′w)(σ′t, σ′u, σ′w, σ′o, σ′e).
  9. σ′e = team(σ′e)(σ′t, σ′u, σ′w, σ′o, σ′e).
  10. σ′w = work(σ′w)(σ′t, σ′u, σ′w, σ′o, σ′e).
  11. if objective(σ′t, σ′u, σ′w, σ′o, σ′e) then ameliorate(σ′t, σ′u, σ′w, σ′o, σ′e) else enterprise(σ′t, σ′u, σ′w, σ′o, σ′e).
end end

0. Σ is a further undefined and unexplained enterprise state space. The various enterprise players view this state in their own way.
1. Six staff group operations, str, tac, opr, sup, tea and wrk, each act in the enterprise state such as conceived by respective groups to effect a resulting enterprise state such as achieved by respective groups.
2. Six staff group state amelioration functions, ame.s, ame.t, ame.o, ame.u, ame.e and ame.w, each apply to the resulting enterprise states such as achieved by respective groups to yield a result state such as achieved by that group.
3. An overall objective function tests whether a state summary reflects that the objectives of the enterprise have been achieved or not.
4. The enterprise function applies to the tuple of six group-biased (i.e., ameliorated) states. Initially these may all be the same state. The result is an ameliorated state.
5. An iteration, that is, a step of enterprise activities, lines 5.–13. proceeds as follows:
6. strategic management operates
   • in its state space, σt: Σ;
   • effects a next (un-ameliorated strategic management) state σ′t;
   • and ameliorates this latter state in the context of all the other player’s ameliorated result states.

7.–11. The same actions take place, simultaneously for the other players: tac, opr, sup, tea and wrk.
12. A test, has objectives been met, is made on the six ameliorated states.
13. If test is successful, then the enterprise terminates in an ameliorated state.
```
14. Otherwise the enterprise recurses, that is, “repeats” itself in new states.

The above “function” definition is suggestive. It suggests that a solution to the fix-point 6-tuple of equations over “intermediate” states, \( \sigma' x \), where \( x \) is any of \( s, t, o, e, w, \) is achievable by iteration over just these 6 equations.

### 7.7.2 Requirements

Top-level, including strategic management tends to not be amenable to “automation”. Increasingly tactical management tends to “divide” time between “bush-fire, stop-gap” actions – hardly automatable and formulating, initiating and monitoring main operations. The initiation and monitoring of tactical actions appear amenable to partial automation. Operational management – with its reliance on rules & regulations, scripts and licenses – is where computer monitoring and partial control has reaped the richest harvests.

### 7.7.3 On Modeling Management and Organisation

Management and organisation basically spans entity, function, event and behaviour intensities and thus typically require the full spectrum of modeling techniques and notations — summarised in Sect. 7.2.3.

### 7.8 Human Behaviour

- **By domain human behaviour** we shall understand any of a quality spectrum of carrying out assigned work: from (i) careful, diligent and accurate, via (ii) sloppy dispatch, and (iii) delinquent work, to (iv) outright criminal pursuit.

Although we otherwise do not go into any depth with respect to the analysis & description of humans, we shall momentarily depart from this “abstinence”.

#### 7.8.1 Conceptual Analysis

To model human behaviour “smacks” like modeling human actors, the psychology of humans, etc. ! We shall not attempt to model the psychological side of humans — for the simple reason that we neither know how to do that nor whether it can at all be done. Instead we shall be focusing on the effects on non-human manifest entities of human behaviour.

---

**Banking — or Programming — Staff Behaviour**

**Example 107** Let us assume a bank clerk, “in ye olde” days, when calculating, say mortgage repayments (cf. Example 96). We would characterise such a clerk as being **diligent**, etc., if that person carefully follows the mortgage calculation rules, and checks and double-checks that calculations “tally up”, or lets others do so. We would characterise a clerk as being **sloppy** if that person occasionally forgets the checks alluded to above. We would characterise a clerk as being **delinquent** if that person systematically forgets these checks. And we would call such a person a **criminal** if that person intentionally miscalculates in such a way that the bank (and/or the mortgage client) is cheated out of funds which, instead, may be diverted to the cheater. Let us, instead of a bank clerk, assume a software programmer charged with implementing an automatic routine for effecting mortgage repayments (cf. Example 96). We would characterise the programmer as being **diligent** if that person carefully follows the mortgage calculation rules, and throughout the development verifies and tests that the calculations are correct with respect to the rules. We would characterise the programmer as being **sloppy** if that person forgets certain checks and tests when otherwise correcting the computing program under development. We would characterise the programmer as being **delinquent** if that person systematically forgets these checks.
and tests. And we would characterise the programmer as being a **criminal** if that person intentionally provides a program which miscalculates the mortgage interest, etc., in such a way that the bank (and/or the mortgage client) is cheated out of funds.

---

**A Human Behaviour Mortgage Calculation**

**Example 108**  Example 96 on Page 173 gave a semantics to the mortgage calculation request (i.e., command) as would a diligent bank clerk be expected to perform it. To express, that is, to model, how sloppy, delinquent, or outright criminal persons (staff?) could behave we must modify the int\(_\text{Cmd}\)(mkPM(c,a,m,p,d))\((\rho,\alpha,\mu,\ell)\) definition.

\[
\text{int}_\text{Cmd}(\text{mkPM}(c,a,m,p,d))(\rho,\alpha,\mu,\ell) \equiv \\
\text{let } (b,d') = \ell(m) \text{ in} \\
\text{if } q(\alpha(a),p) \left[ \alpha(a) \leq p \vee \alpha(a) = p \vee \alpha(a) \leq p \ldots \right] \\
\text{then} \\
\text{let } l = f_1(\text{interest}(m,b,\text{period}(d,d'))), \\
\ell' = \ell \uplus [ m \mapsto f_2(l(m) - (p-i)) ], \\
\alpha' = \alpha \uplus [ a \mapsto f_3(\alpha(a) - p), a \mapsto f_4(\alpha(a) + i), a \mapsto f_{\text{staff}}(\alpha(a_{\text{staff}}) + i) ] \text{ in} \\
((\rho,\alpha',\mu,\ell'),\text{ok}) \text{ end} \\
\text{else} \\
((\rho,\alpha',\mu,\ell),\text{nok}) \text{ end end} \\
\text{pre } c \in \text{dom } \mu \land m \in \mu(c) \\
\text{post } \theta \set = \alpha(\theta) \land \text{action}(s_y,\theta) \in \theta \set \\
\land \forall \theta': \theta \cup \theta' \in \theta \set \Rightarrow \\
\exists \se: \text{RUL-RUL} \in \text{hum}_\text{int}(s_y,\theta) \Rightarrow \se(\theta,\theta')
\]

The predicate \(q\) and the functions \(f_1, f_2, f_3, f_4\) and \(f_{\text{staff}}\) of Example 108 are deliberately left undefined. They are being defined by the “staffer” when performing (incl., programming) the mortgage calculation routine. The point of Example 108 is that one must first define the mortgage calculation script precisely as one would like to see the diligent staff (programmer) to perform (incl., correctly program) it before one can “pinpoint” all the places where lack of diligence may “set in”. The invocations of \(q, f_1, f_2, f_3, f_4\) and \(f_{\text{staff}}\) designate those places. The point of Example 108 is also that we must first domain-define, “to the best of our ability” all the places where human behaviour may play other than a desirable role. If we cannot, then we cannot claim that some requirements aim at countering undesirable human behaviour.

Commensurate with the above, humans interpret rules and regulations differently, and, for some humans, not always consistently — in the sense of repeatedly applying the same interpretations. Our final specification pattern is therefore:

**type**

\[
\text{Action} = \Theta \rightarrow \Theta\text{-infset}
\]

**value**

\[
\text{hum}_\text{int}: \text{Rule} \rightarrow \Theta \rightarrow \text{RUL}\text{-infset} \\
\text{action}: \text{Stimulus} \rightarrow \Theta \rightarrow \Theta \\
\text{hum}_\text{beha}: \text{Stimulus} \times \text{Rules} \rightarrow \text{Action} \rightarrow \Theta \rightarrow \Theta\text{-infset} \\
\text{hum}_\text{beha}(s_y, s_y, \theta) = \theta \set
\]

**post**

\[
\theta \set = \alpha(\theta) \land \text{action}(s_y,\theta) \in \theta \set \\
\land \forall \theta': \theta \cup \theta' \in \theta \set \Rightarrow \\
\exists \se: \text{RUL-RUL} \in \text{hum}_\text{int}(s_y,\theta) \Rightarrow \se(\theta,\theta')
\]
The above is, necessarily, sketchy: There is a possibly infinite variety of ways of interpreting some rules. A human, in carrying out an action, interprets applicable rules and chooses one which that person believes suits some (professional, sloppy, delinquent or criminal) intent. “Suits” means that it satisfies the intent, i.e., yields true on the pre/post-configuration pair, when the action is performed — whether as intended by the ones who issued the rules and regulations or not. We do not cover the case of whether an appropriate regulation is applied or not.

The above-stated axioms express how it is in the domain, not how we would like it to be. For that we have to establish requirements.

7.8.2 Requirements

Requirements in relation to the human behaviour facet is not requirements about software that “replaces” human behaviour. Such requirements were hinted at in Sects. 7.5.2–7.7.2. Human behaviour facet requirements are about software that checks human behaviour; that its remains diligent; that it does not transgress into sloppy, delinquent, let alone criminal behaviour. When transgressions are discovered, appropriate remedial actions may be prescribed.

7.8.3 On Modeling Human Behaviour

To model human behaviour is, “initially”, much like modeling management and organisation. But only ‘initially’. The most significant human behaviour modeling aspect is then that of modeling non-determinism and looseness, even ambiguity. So a specification language which allows specifying non-determinism and looseness (like CafeOBJ [157] and RSL [176]) is to be preferred. To prescribe requirements is to prescribe the monitoring of the human input at the computer interface.

7.9 Summary

7.9.1 Method Principles, Techniques and Tools

Recall that by a method we shall understand a set of principles for selecting and applying a set of techniques using a set of tools in order to construct an artefact.

7.9.1.1 Principles of Modelling Domain Facets

We shall just point out one applied principle, that of:

Conservative Extension.\footnote{We remind the reader of the definition of the concept of conservative extension: An extension of a logical theory is conservative, i.e., conserves, if every theorem expressible in the original theory is also derivable within the original theory [en.wiktionary.org/wiki/conservative_extension] \cite{46, 7]. See also \cite{341, 144, 100, 20, 243, 161} and en.m.wikipedia.org/wiki/Extension_by_new_constant_and_function_names} This principle of making sure that additional domain descriptions form conservative extensions are applied throughout this chapter:

- Support Technologies, Sect. 7.3,
- Rules & Regulations, Sect. 7.4,
- Scripts, Sect. 7.5,
- License Languages, Sect. 7.6,
- Management & Organisation, Sect. 7.7 and
- Human Behaviour, Sect. 7.8.

The concepts of these six additional facets builds upon, i.e., extends, those of Intrinsics, that is, those of Chapters 3–6.

Most of the principles mentioned in earlier chapters have also been applied.
7.9.1.2 Techniques of Modelling Domain Facets

We have already mentioned techniques that have been applied in this chapter’s “On Modelling ...” sections: Sects. 7.2.3 on Page 167, 7.3.3 on Page 171, 7.4.3 on Page 173, 7.5.3 on Page 175, 7.6.5 on Page 184, 7.7.3 on Page 188 and 7.8.3 on the facing page. And shall leave it at that.

7.9.1.3 Tools of Modelling Domain Facets

The tools for modelling, i.e., analysing & describing domain facets have already been mentioned in this chapter's seven sections on the individual facets.

7.9.2 General Issues

7.9.2.1 Completion

Domain acquisition results in typically up to thousands of units of domain descriptions. Domain analysis subsequently also serves to classify which facet any one of these description units primarily characterises. But some such “compartmentalisations” may be difficult, and may be deferred till the step of “completion”. It may then be, “at the end of the day”, that is, after all of the above facets have been modeled that some description units are left as not having been described, not deliberately, but “circumstantially”. It then behooves the domain engineer to fit these “dangling” description units into suitable parts of the domain description. This “slotting in” may be simple, and all is fine. Or it may be difficult. Such difficulty may be a sign that the chosen model, the chosen description, in its selection of entities, functions, events and behaviours to model — in choosing these over other possible selections of phenomena and concepts is not appropriate. Another attempt must be made. Another selection, another abstraction of entities, functions, etc., may need be chosen. Usually however, after having chosen the abstractions of the intrinsic phenomena and concepts, one can start checking whether “dangling” description units can be fitted in “with ease”.

7.9.2.2 Integrating Formal Descriptions

We have seen that to model the full spectrum of domain facets one needs not one, but several specification languages. No single specification language suffices. It seems highly unlikely and it appears not to be desirable to obtain a single, “universal” specification language capable of “equally” elegantly, suitably abstractly modeling all aspects of a domain. Hence one must conclude that the full modeling of domains shall deploy several formal notations — including plain, good old mathematics in all its forms. The issues are then the following which combinations of notations to select, and how to make sure that the combined specification denotes something meaningful. The ongoing series of “Integrating Formal Methods” conferences [10] is a good source for techniques, compositions and meanings.

7.9.2.3 The Impossibility of Describing Any Domain Completely

Domain descriptions are, by necessity, abstractions. One can never hope for any notion of complete domain descriptions. The situation is no better for domains such as we define them than for physics. Physicists strive to understand the manifest world around us – the world that was there before humans started creating “their domains”. The physicists describe the physical world “in bits and pieces” such that large collections of these pieces “fit together”, that is, are based on some commonly accepted laws and in some commonly agreed mathematics. Similarly for such domains as will be the subject of domain science & engineering such as we cover that subject in [78, 66] and in the present chapter and reports [75, 68]. Individual such domain descriptions will be emphasizing some clusters of facets, others will be emphasizing other aspects.

7.9.2.4 Rôles for Domain Descriptions

We can distinguish between a spectrum of rôles for domain descriptions. Some of the issues brought forward below may have been touched upon in [78, 66].

July 6, 2020, 10:03, A Foundation for Software Development
© Dines Bjørner 2020, Fredsvej 11, DK–2840 Holte, Denmark
7.9.2.4.1 Alternative Domain Descriptions:

It may very well be meaningful to avail oneself of a variety of domain models (i.e., descriptions) for any one domain, that is, for what we may consider basically one and the same domain. In control theory (a science) and automation (an engineering) we develop specific descriptions, usually on the form of a set of differential equations, for any one control problem. The basis for the control problem is typically the science of mechanics. This science has many renditions (i.e., interpretations). For the control problem, say that of keeping a missile carried by a train wagon, erect during train movement and/or windy conditions, one may then develop a “self-contained” description of the problem based on some mechanics theory presentation. Similarly for domains. One may refer to an existing domain description. But one may re-develop a textually “smaller” domain description for any one given, i.e., specific problem.

7.9.2.4.2 Domain Science:

A domain description designates a domain theory. That is, a bundle of propositions, lemmas and theorems that are either rather explicit or can be proven from the description. So a domain description is the basis for a theory as well as for the discovery of domain laws, that is, for a domain science. We have sciences of physics (incl. chemistry), biology, etc. Perhaps it is about time to have proper sciences, to the extent one can have such sciences for human-made domains.

7.9.2.4.3 Business Process Re-engineering:

Some domains manifest serious amounts of human actions and interactions. These may be found to not be efficient to a degree that one might so desire. A given domain description may therefore be a basis for suggesting other management & organisation structures, and/or rules & regulations than present ones. Yes, even making explicit scripts or a license language which have hitherto been tacitly understood – without necessarily computerising any support for such a script or license language. The given and the resulting domain descriptions may then be the basis for operations research models that may show desired or acceptable efficiency improvements.

7.9.2.4.4 Software Development:

[66] shows one approach to requirements prescription. Domain analysis & description, i.e., domain engineering, is here seen as an initial phase, with requirements prescription engineering being a second phase, and software design being a third phase. We see domain engineering as indispensable, that is, an absolute must, for software development. [58, Domains: Their Simulation, Monitoring and Control] further illustrates how domain engineering is a base for the development of domain simulators, demos, monitors and controllers.

7.9.2.5 Grand Challenges of Informatics

To establish a reasonably trustworthy and believable theory of a domain, say the transportation, or just the railway domain, may take years, possibly 10–15! Similarly for domains such as the financial service industry, the market (of consumers and producers, retailers, wholesaler, distribution cum supply chain), health care, and so forth. The current author urges younger scientists to get going! It is about time.

7.10 Bibliographical Notes

To create domain descriptions, or requirements prescriptions, or software designs, properly, at least such as this author sees it, is a joy to behold. The beauty of carefully selected and balanced abstractions, their interplay with other such, the relations between phases, stages and steps, and many more conceptual constructions make software engineering possibly the most challenging intellectual pursuit today. For this and more consult [39, 40, 41].

16 In the early-to-mid 2000s there were a rush of research foundations and scientists enumerating “Grand Challenges of Informatics”
7.11 Exercise Problems

7.11.1 Research Problems

Exercise 38 A Research Challenge. Mathematical Explanation: The seven facets identified in this chapter are not identified with respect to one another on the basis of some mathematical model. In what [theoretical] computer science sense could one hold them out from one another? If you have a solution please present it.

Exercise 39 A Research Challenge. Other Facets?: In this chapter we have identified six facets beyond the intrinsics. The research challenge here is to identify more facets and to give them a treatment like in this chapter or even as suggested in the above exercise.

7.11.2 Term Projects

We continue the term projects of Sects. 3.24.3 on Page 81, 4.12.3 on Page 121 and 6.14.3 on Page 161. The students are to identify and analyse & describe at least three distinct facets of their chosen domain, that is:

- variations of intrinsics,
- support technology,
- rules & regulations,
- scripts,
- license language,
- management & organisation,
- human behaviour.

Exercise 40 An MSc Student Exercise. The Consumer Market, Facets: We refer to Exercises 4 on Page 82, 20 on Page 121, 31 on Page 161 and 40.

Exercise 41 An MSc Student Exercise. Financial Service Industry, Facets: We refer to Exercises 5 on Page 82, 21 on Page 121 and 32 on Page 161.

Exercise 42 An MSc Student Exercise. Container Line Industry, Facets: We refer to Exercises 6 on Page 82, 22 on Page 121, and 33 on Page 161.

Exercise 43 An MSc Student Exercise. Railway Systems, Facets: We refer to Exercises 7 on Page 82, 23 on Page 121, and 34 on Page 161.

Exercise 44 A PhD Student Problem. Part-Material Conjoins: Canals, Facets: We refer to Exercises 8 on Page 82, 24 on Page 121 and 35 on Page 161.


These exercise problems are continued in Sect. 8.9.2 on Page 241.
Part III

REQUIREMENTS
In this chapter we show one approach to systematically, but, of course, not automatically, “derive” requirements prescriptions from domain descriptions. We shall introduce and treat quite a vocabulary of concepts (i) machine; (ii-iv) domain, interface and machine requirements; (v-ix) projection, instantiation, determination, extension and fitting; and (x) derived requirements. The approach we show is novel [44]. It does not replace conventional requirements engineering [355]. Merely supplements it. The conventional approach is not founded on domain descriptions, although frequent references are made, more-or-less implicitly, to domains. We therefore find it justified to present the view of this chapter that requirement prescriptions can be rather systematically arrived at from a series of analyses & rewritings of domain descriptions.

8.1 Introduction

8.1.1 The Contribution of this Chapter

We claim that the present chapter contributes to our understanding and practice of software engineering as follows: (1) it shows how the new phase of engineering, domain engineering, forms a prerequisite for requirements engineering; (2) it endows the “classical” form of requirements engineering with a structured set of development stages and steps: (a) first a domain requirements stage, (b) to be followed by an interface requirements stage, and (c) to be concluded by a machine requirements stage; (3) it further structures and gives a reasonably precise contents to the stage of domain requirements: (i) first a projection step, (ii) then an instantiation step, (iii) then a determination step, (iv) then an extension step, and (v) finally a fitting step — with these five steps possibly being iterated; and (4) it also structures and gives a reasonably precise contents to the stage of interface requirements based on a notion of shared entities. Each of the steps (i–v) open for the possibility of simplifications. Steps (a–c) and (i–v), we claim, are new. They reflect a serious contribution, we claim, to a logical structuring of the field of requirements engineering and its very many otherwise seemingly diverse concerns.

8.1.2 Some Comments

This chapter is, perhaps, unusual in the following respects: (i) It is a methodology chapter, hence there are no “neat” theories about development, no succinctly expressed propositions, lemmas nor theorems, and hence no proofs. (ii) As a consequence the chapter is borne by many, and by extensive examples. (iii) The examples of this chapter are all focused on a generic road transport net. (iv) To reasonably fully exemplify the requirements approach, illustrating how our method copes with a seeming complexity of interrelated method aspects, the full example of this chapter embodies very many description and prescription elements: hundreds of concepts (types, axioms, functions). (v) This methodology chapter covers a “grand” area of software engineering: Many textbooks and papers are written on Requirements Engineering. We postulate, in contrast to all such books (and papers), that requirements engineering should be founded on domain engineering. Hence we must, somehow, show that our approach relates to major elements of what the Requirements Engineering books put forward. (vi) As a result, this chapter is long.

---

1 — where these proofs would be about the development theories. The example development of requirements do imply properties, but formulation and proof of these do not constitute new contributions — so are left out.
8.1.3 Structure of Chapter

The structure of the chapter is as follows: Section 8.2 provides a fair-sized, hence realistic example. Sections 8.3–8.5 covers our approach to requirements development. Section 8.3 overviews the issue of “requirements”; relates our approach (i.e., Sects. 8.4–8.5) to systems, user and external equipment and functional requirements; and Sect. 8.3 also introduces the concepts of the machine to be requirements prescribed, the domain, the interface and the machine requirements. Section 8.4 covers the domain requirements stages of projection (Sect. 8.4.1), instantiation (Sect. 8.4.2), determination (Sect. 8.4.3), extension (Sect. 8.4.4) and fitting (Sect. 8.4.5). Section 8.5 covers key features of interface requirements: shared phenomena (Sect. 8.5.1.1), shared endurants (Sect. 8.5.1.2) and shared actions, shared eventand shared behaviours (Sect. 8.5.1.3). Section 8.5.1.3 further introduces the notion of derived requirements. Section 8.7 concludes the chapter.

8.2 An Example Domain: Transport

We refer to the “Running Example” of Chapters 3–6

Chapters 3–6 brought a consolidated version of the “running” road transport system example

In order to exemplify the various stages and steps of requirements development we first bring a domain description example. The example follows the steps of an idealised domain description. First we describe the endurants, then we describe the perdurants. Endurant description initially focus on the composite and atomic parts. Then on their “internal” qualities: unique identifications, mereologies, and attributes. The descriptions alternate between enumerated, i.e., labeled narrative sentences and correspondingly “numbered” formalisations. The narrative labels cum formula numbers will be referred to, frequently in the various steps of domain requirements development.

8.2.1 Endurants

Since we have chosen a manifest domain, that is, a domain whose endurants can be pointed at, seen, touched, we shall follow the analysis & description process as outlined in [70] and formalised in [62]. That is, we first identify, analyse and describe (manifest) parts, composite and atomic, abstract (Sect. 8.2.2) or concrete (Sect. 8.2.2.1). Then we identify, analyse and describe their unique identifiers (Sect. 8.2.2.2), mereologies (Sect. 8.2.2.3), and attributes (Sects. 8.2.2.4–8.2.2.4).

The example fragments will be presented in a small type-font.

8.2.2 Domain, Net, Fleet and Monitor

The root domain, $\Delta$, is that of a composite traffic system (286a.) with a road net, (286b.) with a fleet of vehicles and (286c.) of whose individual position on the road net we can speak, that is, monitor.$^{3}$

$^2$ The example of this section is that of the “running example” of Chapters 3–6

$^3$ The monitor can be thought of, i.e., conceptualised. It is not necessarily a physically manifest phenomenon.

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>286</td>
<td>$\Delta$</td>
</tr>
<tr>
<td>286a</td>
<td>obs$_N$: $\Delta \rightarrow N$</td>
</tr>
<tr>
<td>286b</td>
<td>obs$_F$: $\Delta \rightarrow F$</td>
</tr>
<tr>
<td>286c</td>
<td>obs$_M$: $\Delta \rightarrow M$</td>
</tr>
</tbody>
</table>

We analyse the traffic system into

- a composite road net,
- a composite fleet (of vehicles), and
- an atomic monitor.
The road net consists of two composite parts, an aggregation of hubs and an aggregation of links.

**Type and Value**

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>287a HA</td>
<td>obs_HA: N → HA</td>
</tr>
<tr>
<td>287b LA</td>
<td>obs_LA: N → LA</td>
</tr>
</tbody>
</table>

### 8.2.2.1 Hubs and Links

Hub aggregates are sets of hubs. Link aggregates are sets of links.

**Type and Value**

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>288 H, HS = H-set</td>
<td>obs_HS: HA → HS</td>
</tr>
<tr>
<td>289 L, LS = L-set</td>
<td>obs_LS: LA → LS</td>
</tr>
<tr>
<td>290 V, VS = V-set</td>
<td>obs_VS: F → VS</td>
</tr>
</tbody>
</table>

We introduce some auxiliary functions.

- **links**: extracts the links of a network.
- **hubs**: extracts the hubs of a network.

**Value**

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>291 links(δ)</td>
<td>obs_LA(obs_N(δ))</td>
</tr>
<tr>
<td>291 hubs(δ)</td>
<td>obs_HS(obs_HA(obs_N(δ)))</td>
</tr>
</tbody>
</table>

### 8.2.2.2 Unique Identifiers

Applying `observe_unique_identifier`, the domain description prompt 9 on Page 86, to the observed parts yields the following.

Nets, hub and link aggregates, hubs and links, fleets, vehicles and the monitor all have unique identifiers such that all such are distinct, and with corresponding observers.

**Type and Value**

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>292a NI, HAI, LAI, HI, LI, FI, VI, MI</td>
<td>uid_LI: L → LI</td>
</tr>
<tr>
<td>292b uid_HI: H → HI</td>
<td>uid_MI: M → MI</td>
</tr>
<tr>
<td>292c uid_NI: N → NI</td>
<td>uid_FL: F → FI</td>
</tr>
<tr>
<td>292d uid_HAI: HA → HAI</td>
<td>uid_VI: V → VI</td>
</tr>
</tbody>
</table>

Axiom 292b is expressed semi-formally, in mathematics. We introduce some auxiliary functions:

- **xtr_lis**: extracts all link identifiers of a traffic system.
- **xtr_his**: extracts all hub identifiers of a traffic system.
- **get_link**: retrieves the designated link.
- **get_hub**: retrieves the designated hub.
8.2.2.3 Mereology

We cover the mereologies of all part sorts introduced so far. We decide that nets, hub aggregates, link aggregates and fleets have no mereologies of interest. Applying observe mereology, the domain description prompt 10 on Page 90, to hubs, links, vehicles and the monitor yields the following.

297 Hub mereologies reflect that they are connected to zero, one or more links.
298 Link mereologies reflect that they are connected to exactly two distinct hubs.
299 Vehicle mereologies reflect that they are connected to the monitor.
300 The monitor mereology reflects that it is connected to all vehicles.
301 For all hubs of any net it must be the case that their mereology designates links of that net.
302 For all links of any net it must be the case that their mereology designates hubs of that net.
303 For all transport domains it must be the case that
   a the mereology of vehicles of that system designates the monitor of that system, and that
   b the mereology of the monitor of that system designates vehicles of that system.

value
297 \[\text{obs.mereo}_H : H \rightarrow \text{Li-set}\]
298 \[\text{obs.mereo}_L : L \rightarrow \text{Hi-set}\]

axiom
298 \(\forall l : \text{L-card} \, \text{mereo}_L(l) = 2\)

value
299 \[\text{obs.mereo}_V : V \rightarrow \text{Mi}\]
300 \[\text{obs.mereo}_M : M \rightarrow \text{Vi-set}\]

axiom
301 \(\forall \delta : \Delta, hs : \text{HS} \, hs = \text{hubs(\delta)}, ls : \text{LS} \, ls = \text{links(\delta)} \cdot\)
301 \(\forall h : H \, h \in \text{hs.mereo}_H(h) \subseteq \text{xtr.lis(\delta)} \land\)
302 \(\forall l : L \, l \in \text{ls.mereo}_L(l) \subseteq \text{xtr.his(\delta)} \land\)
303a \(\text{let } f : F = \text{obs}_F(\delta) \Rightarrow\)
303a \(\text{let } m : M = \text{obs}_M(\delta), vs : \text{VS} = \text{obs}_V(\delta) \in\)
303a \(\forall v : V \, v \in vs \Rightarrow uid_V(v) \in \text{mereo}_M(m) \land \text{mereo}_M(m) = \{uid_V(v) | v : V, v \in vs\}\)
303b end end

8.2.2.4 Attributes, I

We may not have shown all of the attributes mentioned below — so consider them informally introduced!

- **Hubs**: locations are considered static, hub states and hub state spaces are considered programmable;
- **Links**: lengths and locations are considered static, link states and link state spaces are considered programmable;
- **Vehicles**: manufacturer name, engine type (whether diesel, gasoline or electric) and engine power (kW/horse power) are considered static; velocity and acceleration may be considered reactive (i.e., a function of gas pedal position, etc.), global position (informed via a GNSS: Global Navigation Satellite System) and local position (calculated from a global position) are considered biddable.

4 By location we mean a geodetic position.
Applying \texttt{observe\_attributes}, the domain description prompt 11 on Page 95, to hubs, links, vehicles and the monitor yields the following.

First hubs.

304 Hubs
   a have geodetic locations, \texttt{GeoH}, 
   b have hub states which are sets of pairs of identifiers of links connected to the hub\textsuperscript{5},
   c and have hub state spaces which are sets of hub states\textsuperscript{6}.

305 For every net,
   a link identifiers of a hub state must designate links of that net, and
   b every hub state of a net must be in the hub state space of that hub.

306 We introduce an auxiliary function: \texttt{xtr\_lis} extracts all link identifiers of a hub state.

\begin{itemize}
\item \texttt{type} \texttt{GeoH}
\item \texttt{H\Sigma} = (LI\times LI)-set
\item \texttt{H\Omega} = H\Sigma-set
\end{itemize}

\begin{itemize}
\item value
\item \texttt{attr\_GeoH: H \rightarrow GeoH}
\item \texttt{attr\_H\Sigma: H \rightarrow H\Sigma}
\item \texttt{attr\_H\Omega: H \rightarrow H\Omega}
\end{itemize}

\begin{itemize}
\item axiom
\item \forall \delta: \Delta \cdot \texttt{let} \texttt{hs} = \texttt{hubs(\delta)} \texttt{in}
\item \forall h:H \cdot h \in \texttt{hs} \cdot \texttt{xtr\_lis(h)} \subseteq \texttt{xtr\_lis(\delta)}
\item \forall h:H \cdot \texttt{attr\_\Sigma(h)} \subseteq \texttt{attr\_\Omega(h)}
\end{itemize}

\begin{itemize}
\item value
\item \texttt{xtr\_lis: H \rightarrow LI-set}
\item \texttt{xtr\_lis(h)} \equiv \{li | li,li',li'':LI \times LI \cdot (li',li'') \in \texttt{attr\_\Sigma(h)} \land li \in \{li',li''\}\}
\end{itemize}

Then links.

307 Links have lengths.

308 Links have geodetic location.

309 Links have states and state spaces:
   a States modeled here as pairs, \((h\prime,h\prime\prime)\), of identifiers the hubs with which the links are connected and indicating directions (from hub \(h\prime\) to hub \(h\prime\prime\)). A link state can thus have 0, 1, 2, 3 or 4 such pairs.
   b State spaces are the set of all the link states that a link may enjoy.

\begin{itemize}
\item \texttt{type}
\item \texttt{LEN}
\item \texttt{GeoL}
\item \texttt{L\Sigma} = (HI\times HI)-set
\item \texttt{L\Omega} = L\Sigma-set
\end{itemize}

\begin{itemize}
\item value
\item \texttt{attr\_LEN: L \rightarrow LEN}
\item \texttt{attr\_GeoL: L \rightarrow GeoL}
\item \texttt{attr\_L\Sigma: L \rightarrow L\Sigma}
\item \texttt{attr\_L\Omega: L \rightarrow L\Omega}
\end{itemize}

\begin{itemize}
\item axiom
\item \forall n:N \cdot \texttt{let} \texttt{ls = xtr\_links(n), hs = xtr\_hubs(n)} \texttt{in}
\end{itemize}

\textsuperscript{5} A hub state “signals” which input-to-output link connections are open for traffic.

\textsuperscript{6} A hub state space indicates which hub states a hub may attain over time.
REQUIREMENTS

∀ l:L ∈ ls ⇒
∀ σ = attr_LΣ(l) in
0 ≤ card σ ≤ 4
∀ (hi',hi''):((Hi×Hi)∪(hi',hi'')) ∈ lσ ⇒ {hi',hi''} = mereo_L(l)
∀ (hi',hi'') ∈ attr_LΩ(l) end

Then vehicles.

Every vehicle of a traffic system has a position which is either ‘on a link’ or ‘at a hub’.

a An ‘on a link’ position has four elements: a unique link identifier which must designate a link of that traffic system and a pair of unique hub identifiers which must be those of the mereology of that link.
b The ‘on a link’ position real is the fraction, thus properly between 0 (zero) and 1 (one) of the length from the first identified hub “down the link” to the second identifier hub.
c An ‘at a hub’ position has three elements: a unique hub identifier and a pair of unique link identifiers — which must be in the hub state.

distribute takes a net and a set of vehicles and generates a map from vehicles to distinct vehicle positions on the net.

We sketch a “formal” distribute function, but, for simplicity we omit the technical details that secures distinctness — and leave that to an axiom!

We define two auxiliary functions:

xtr_links extracts all links of a net and
xtr_hub extracts all hubs of a net.
And finally monitors. We consider only one monitor attribute.

a For every vehicle of the road transport system the vehicle traffic attribute records a possibly empty list of time marked vehicle positions.

b These vehicle positions are alternate sequences of ‘on link’ and ‘at hub’ positions

i such that any sub-sequence of ‘on link’ positions record the same link identifier, the same pair of ‘to’ and ‘from’ hub identifiers and increasing fractions,

ii such that any sub-segment of ‘at hub’ positions are identical,

iii such that vehicle transition from a link to a hub is commensurate with the link and hub mereologies, and

iv such that vehicle transition from a hub to a link is commensurate with the hub and link mereologies.

type
313 Traffic = VI \rightarrow (T \times VPos)^* 

value
313 attrTraffic: M \rightarrow Traffic

axiom
313b ∀ δ: ∆ •
313b let m = obsM(δ) in
313b let tf = attrTraffic(m) in
313b dom tf ⊆ xtrvis(δ) ∧
313b ∀ vi:VI • vi ∈ dom tf •
313b let tr = tf(vi) in
313b dom tr ⊆ xtrlis(δ) \cup \{fhi,thi\} = mereoL(getLink(li)(δ)),
313b (li,thi) ∈ mereoH(getHub(hi)(δ)),
313b (li,thi) ∈ mereoL(getLink(li)(δ)),
313b (li,thi) ∈ mereoH(getHub(hi)(δ))

313(b)ii (atH(hi,fli,tli),onL(li,fhi,thi,1))

+ case (vp,vp') of
313(b)ii (onL(li,fhi,thi,r),onL(li',fhi',thi',r'))

313(b)ii (atH(hi,fli,thi,0))

et cetera,
313b end end end end end

8.2.3 Perdurants

Our presentation of example perdurants is not as systematic as that of example endurants. Give the simple basis of endurants covered above there is now a huge variety of perdurants, so we just select one example from each of the three classes of perdurants (as outline in [70]): a simple hub insertion action (Sect. 8.2.3.1), a simple link disappearance event (Sect. 8.2.3.2) and a not quite so simple behaviour, that of road traffic (Sect. 8.2.3.3).
8.2.3.1 Hub Insertion Action

Initially inserted hubs, \( h \), are characterised

- by their unique identifier which not one of any hub in the net, \( n \), into which the hub is being inserted,
- by a mereology, \( \{\} \), of zero link identifiers, and
- by — whatever — attributes, \( \text{attrs} \), are needed.

The result of such a hub insertion is a net, \( n' \),

- whose links are those of \( n \), and
- whose hubs are those of \( n \) augmented with \( h \).

\[
\begin{align*}
\text{value} & \quad \text{insert}_\text{hub} : H \to N \to N \\
\text{pre} & \quad \text{uid}_H(h) \notin \text{xtr}_\text{his}(n) \\
\text{obs}_\text{mero}_H & = \{\} \\
\text{post} & \quad \text{obs}_\text{Ls}(n) = \text{obs}_\text{Ls}(n') \\
& \quad \text{obs}_\text{Hs}(n) \cup \{h\} = \text{obs}_\text{Hs}(n')
\end{align*}
\]

8.2.3.2 Link Disappearance Event

We formalise aspects of the link disappearance event:

- The result net, \( n:N' \), is not well-formed.
- For a link to disappear there must be at least one link in the net;
- and such a link may disappear such that
- it together with the resulting net makes up for the “original” net.

\[
\begin{align*}
\text{value} & \quad \text{link}_\text{diss}_\text{event} : N \times N' \times \text{Bool} \\
\text{pre} & \quad \text{obs}_\text{Ls}(\text{obs}_\text{LS}(n)) \neq \{\} \\
\text{post} & \quad \exists l : l \in \text{obs}_\text{Ls}(\text{obs}_\text{LS}(n)) \Rightarrow \\
& \quad l \notin \text{obs}_\text{Ls}(\text{obs}_\text{LS}(n')) \\
& \quad n' \cup \{l\} = \text{obs}_\text{Ls}(\text{obs}_\text{LS}(n))
\end{align*}
\]

8.2.3.3 Road Traffic

The analysis & description of the road traffic behaviour is composed (i) from the description of the global values of nets, links and hubs, vehicles, monitor, a clock, and an initial distribution, \( \text{map} \), of vehicles, “across” the net; (ii) from the description of channels between vehicles and the monitor; (iii) from the description of behaviour signatures, that is, those of the overall road traffic system, the vehicles, and the monitor; and (iv) from the description of the individual behaviours, that is, the overall road traffic system, \( \text{rts} \), the individual vehicles, \( \text{veh} \), and the monitor, \( \text{mon} \).

8.2.3.3.1 Global Values:

There is given some globally observable parts.

- besides the domain, \( \delta : \Delta \),
- a net, \( n:N \),
- a set of vehicles, \( \text{vs}: \text{V-set} \),
- a monitor, \( m:M \), and
An Example Domain: Transport

8.2

A clock, clock, behaviour.

From the net and vehicles we generate an initial distribution of positions of vehicles.

The \( n: N \), \( vs: V \)-set and \( m: M \) are observable from any road traffic system domain \( \delta \).

\[
\begin{align*}
\delta: \Delta \\
n: N &= \text{obs}_N(\delta), \\
ls: L\text{-set} &= \text{links}(\delta), \text{hs: } H\text{-set} = \text{hubs}(\delta), \\
ls: L\text{-set} &= \text{tr}_L(\delta), \text{his: } H\text{-set} = \text{tr}_H(\delta) \\
va: VS &= \text{obs}_V(\text{obs}_F(\delta)), \\
vs: Vs\text{-set} &= \text{obs}_V(\text{obs}_F(\delta)), \\
vis: VI\text{-set} &= \{ \text{uid}_V(\nu) | \nu: V \in vs \}, \\
m: obs_M(\delta), \\
mi &= \text{uid}_M(\mu), \\
ma: attributes(\mu) \\
\text{clock}: T \rightarrow \text{out} \{ \text{clk}_{ch}[\nu|\nu: V\in \text{vs}] \} \text{ Unit} \\
\text{vm: MAP-vpos-map} &= \text{distribute}(\text{vs})(\eta);
\end{align*}
\]

8.2.3.3.2 Channels:

We additionally declare a set of vehicle-to-monitor-channels indexed

\begin{enumerate}
\item by the unique identifiers of vehicles
\item and the (single) monitor identifier.\(^7\)
\end{enumerate}

and communicating vehicle positions.

\[
\begin{align*}
\text{channel} \\
\{ \text{vm}_{ch}[\nu, \mu] | \nu: V\text{-set} = \{ \text{uid}_V(\nu) | \nu: V\in \text{vis} \} \}: V\text{Pos}
\end{align*}
\]

8.2.3.3 Behaviour Signatures:

The road traffic system behaviour, \( \text{rts} \), takes no arguments (hence the first Unit)\(^8\); and “behaves”, that is, continues forever (hence the last Unit).

The vehicle behaviour

\begin{enumerate}
\item is indexed by the unique identifier, \( \text{uid}_V(\nu): V\text{-set} \),
\item the vehicle mereology, in this case the single monitor identifier \( \mu: M\text{-set} \),
\item the vehicle attributes, \( \text{obs}_v\text{attributes}(\nu) \)
\item and — factoring out one of the vehicle attributes — the current vehicle position.
\item The vehicle behaviour offers communication to the monitor behaviour (on channel \( \text{vm}_{ch}[\nu] \)); and behaves “forever”.
\end{enumerate}

The monitor behaviour takes

\begin{enumerate}
\item the monitor identifier,
\item the monitor mereology,
\item the monitor attributes,
\item and — factoring out one of the vehicle attributes — the discrete road traffic, \( \text{drft: } d\text{RTF} \), being repeatedly “updated” as the result of input communications from (all) vehicles;
\item the behaviour otherwise behaves forever.
\end{enumerate}

\[
\begin{align*}
\text{value} \\
\text{rts}: \text{Unit } \rightarrow \text{Unit} \\
\text{veh}_{\nu: V}: \mu: M\text{-set} \rightarrow \text{vp}: V\text{Pos } \rightarrow \text{out} \text{vm}_{ch}[\nu, \mu] \text{ Unit} \\
\text{mon}_{\nu: V}: \text{vis}: V\text{-set} \rightarrow \text{RTF } \rightarrow \text{in} \{ \text{vm}_{ch}[\nu, \mu] | \nu: V\text{-set} \}, \text{clk}_{ch} \text{ Unit}
\end{align*}
\]

\(^7\) Technically speaking: we could omit the monitor identifier.

\(^8\) The Unit designator is an RSL technicality.
Thus we shall consider our road traffic system, \(\text{rts}\), as

- the concurrent behaviour of a number of vehicles and, to “observe”, or, as we shall call it, to monitor their movements,
- the monitor behaviour.

\[
\text{value} \quad \text{rts}() = \begin{align*}
&\parallel \{ \text{veh}_{\text{uid} \cdot \nu}^{\text{r}(v)}(\text{vm}(\text{id}_\text{L}(v))) | v : \text{V} \in \text{vs} \} \\
&\parallel \text{mon}_{\text{mi}}(\text{vis})(| v : \text{vi} \in \text{vis} |)
\end{align*}
\]

where, wrt, the monitor, we dispense with the mereology and the attribute state arguments and instead just have a monitor traffic argument which records the discrete road traffic, \(\text{MAP}\), initially set to “empty” traces (\(\langle\rangle\), of so far “no road traffic”).

In order for the monitor behaviour to assess the vehicle positions these vehicles communicate their positions to the monitor via a vehicle to monitor channel. In order for the monitor to time-stamp these positions it must be able to “read” a clock.

We describe here an abstraction of the vehicle behaviour at a Hub (\(\text{hi}\)).

- Either the vehicle remains at that hub informing the monitor of its position,
- or, internally non-deterministically,
  - moves onto a link, \(\text{li}\), whose “next” hub, identified by \(\text{thi}\), is obtained from the mereology of the link identified by \(\text{li}\);
  - informs the monitor, on channel \(\text{vm}[v_i,mi]\), that it is now at the very beginning (0) of the link identified by \(\text{li}\), whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning of that link,
- or, again internally non-deterministically, the vehicle “disappears — off the radar”!

\[
\text{veh}_{v_i}(mi)(vp : \text{atH}(\text{hi}, \text{fl}, \text{li})) \equiv
\begin{align*}
&\text{v_m_ch}[v_i,mi]!vp ; \text{veh}_{v_i}(mi)(vp) \\
&\text{let } \{ \text{hi}', \text{thi} \} = \text{mero}_L(\text{get_link}(\text{li})(n)) \text{ in}
\end{align*}
\]

\[
\begin{align*}
&\text{assert: } \text{hi}' = \text{hi} \\
&\text{veh}_{v_i}(mi)(\text{onL}(\text{li}, \text{hi}, \text{thi}, 0)) \text{ end}
\end{align*}
\]

Stop

We describe here an abstraction of the vehicle behaviour on a Link (\(\text{li}\)). Either

- the vehicle remains at that link position informing the monitor of its position,
- or, internally non-deterministically, if the vehicle’s position on the link has not yet reached the hub,
  - then the vehicle moves an arbitrary increment \(\ell_e\) (less than or equal to the distance to the hub) along the link informing the monitor of this, or
  - else,
    - while obtaining a “next link” from the mereology of the hub (where that next link could very well be the same as the link the vehicle is about to leave),
    - the vehicle informs the monitor that it is now at the hub identified by \(\text{thi}\), whereupon the vehicle resumes the vehicle behaviour positioned at that hub.
- or, internally non-deterministically, the vehicle “disappears — off the radar”!

\[
\text{veh}_{v_i}(mi)(vp : \text{onL}(\text{li}, \text{hi}, \text{thi}, r)) \equiv
\begin{align*}
&\text{v_m_ch}[v_i,mi]!vp ; \text{veh}_{v_i}(mi, va)(vp) \\
&\text{if } r + \ell_e \leq 1
\end{align*}
\]

\[
\text{then}
\]

\[
\begin{align*}
&\text{veh}_{v_i}(mi)(\text{onL}(\text{li}, \text{hi}, \text{thi}, r + \ell_e)) ;
\end{align*}
\]
332(b)i \[ \text{veh}_{v_0}(\text{mi})(\text{onL}(l_i,\text{fhi},t_i,r+\ell_i)) \]
332(b)ii \[ \text{else} \]
332(b)ii1 \[ \text{let } l_i':\text{LH} \in \text{mer}o_\mathcal{H}(\text{get}_\text{hub}(\text{thi}(n))) \text{ in} \]
332(b)ii2 \[ v_{\text{mch}}[v_i,\text{mi}](\text{latH}(l_i,\text{thi},l_i')); \]
332(b)ii3 \[ \text{veh}_{v_0}(\text{mi})(\text{athH}(l_i,\text{thi},l_i')) \text{ end end} \]
332c \[ \text{stop} \]

**The Monitor Behaviour**

333 The monitor behaviour evolves around

a the monitor identifier,

b the monitor mereology,

c and the attributes, \( ma:\text{ATTR} \)

d — where we have factored out as a separate arguments — a table of traces of time-stamped vehicle positions,

e while accepting messages

i about time

ii and about vehicle positions

f and otherwise progressing “indefinitely”.

334 Either the monitor “does own work”

335 or, internally non-deterministically accepts messages from vehicles.

a A vehicle position message, \( vp \), may arrive from the vehicle identified by \( vi \).

b That message is appended to that vehicle’s movement trace – prefixed by time (obtained from the time channel),

c whereupon the monitor resumes its behaviour —

d where the communicating vehicles range over all identified vehicles.

333 \[ \text{mon}_m(\text{vis})(\text{trf}) \equiv \]
334 \[ \text{mon}_m(\text{vis})(\text{trf}) \]
335 \[ \text{mon}_m(\text{vis})(\text{trf}) \]
335a \[ \{ \text{let } tvp = (\text{clk}_\text{ch}?.,v_{\text{mch}}[v_i,\text{mi}])? \text{ in} \]
335b \[ \text{let } trf' = trf \uplus [v_i \mapsto trf(v_i)\uplus tvb] \text{ in} \]
335c \[ \text{mon}_m(\text{vis})(\text{trf'}) \]
335d \[ \text{end end} \]

We are about to complete a long, i.e., a 6.3 page example (!). We can now comment on the full example:

The domain, \( \delta : \Delta \) is a manifest part. The road net, \( n : N \) is also a manifest part. The fleet, \( f : F \), of vehicles, \( vs : VS \), likewise, is a manifest part. But the monitor, \( m : M \), is a concept. One does not have to think of it as a manifest “observer”. The vehicles are on — or off — the road (i.e., links and hubs). We know that from a few observations and generalise to all vehicles. They either move or stand still. We also, similarly, know that. Vehicles move. Yes, we know that. Based on all these repeated observations and generalisations we introduce the concept of vehicle traffic. Unless positioned high above a road net — and with good binoculars — a single person cannot really observe the traffic. There are simply too many links, hubs, vehicles, vehicle positions and times. Thus we conclude that, even in a richly manifest domain, we can also “speak of”, that is, describe concepts over manifest phenomena, including time!

8.2.4 Domain Facets

The example of this section, i.e., Sect. 8.2, focuses on the domain facet [52, 2008] of (i) intrinsic. It does not reflect the other domain facets: (ii) domain support technologies, (iii) domain rules, regulations & scripts, (iv) organisation & management, and (v) human behaviour. The requirements examples, i.e., the rest of this chapter, thus builds only on the domain intrinsic. This means that we shall not be able to cover principles, technique and tools for the prescription of such important requirements that handle failures of support technology or humans. We shall, however point out where we think such, for example, fault tolerance requirements prescriptions “fit in” and refer to relevant publications for their handling.
8.3 Requirements

This and the next three sections, that is, Sects. 8.4–8.5., are the main sections of this chapter. Section 8.4. is the most detailed and systematic section. It covers the domain requirements operations of projection, instantiation, determination, extension and, less detailed, fitting. Section 8.5. surveys the interface requirements issues of shared phenomena: shared endurants, shared actions, shared events and shared behaviour, and “completes” the exemplification of the detailed domain extension of our requirements into a road pricing system. Section 8.5. also covers the notion of derived requirements. Sections 8.4.–8.5. covers initial requirements. By initial requirements we shall, “operationally” speaking, understand the requirements that are typically derived either from the domain facet descriptions of intrinsic, the support technology, the rules & regulations, the organisation & management, and the human behaviour facets — not covered in this chapter, and/or by more conventional means [134, 253, 380, 264, 256, 301, 355].

Definition: 70 Requirements (I): By a requirements we understand (cf., [245, IEEE Standard 610.12]): “A condition or capability needed by a user to solve a problem or achieve an objective”

The objective of requirements engineering is to create a requirements prescription: A requirements prescription specifies observable properties of endurants and perdurants of the machine such as the requirements stake-holders wish them to be. The machine is what is required: that is, the hardware and software that is to be designed and which are to satisfy the requirements. A requirements prescription thus (putatively) expresses what there should be. A requirements prescription expresses nothing about the design of the possibly desired (required) software. But as the requirements prescription is presented in the form of a model, one can base the design on that model. We shall show how a major part of a requirements prescription can be “derived” from “its” prerequisite domain description.

Rule 1 The “Golden Rule” of Requirements Engineering: Prescribe only those requirements that can be objectively shown to hold for the designed software. ‘Objectively shown’ means that the designed software can either be tested, or be model checked, or be proved (verified), to satisfy the requirements. Caveat: Since we do not illustrate formal tests, model checking nor theorem proving, we shall, alas, not illustrate adherence to this rule.

Rule 2 An “Ideal Rule” of Requirements Engineering: When prescribing (including formalising) requirements, also formulate tests and properties for model checking and theorems whose proof should show adherence to the requirements.

The rule is labelled “ideal” since such precautions will not be shown in this chapter. The rule is clear. It is a question for proper management to see that it is adhered to. See the “Caveat” above!

Rule 3 Requirements Adequacy: Make sure that requirements cover what users expect. That is, do not express a requirement for which you have no users, but make sure that all users’ requirements are represented or somehow accommodated. In other words: the requirements gathering process needs to be like an extremely “fine-meshed net”: One must make sure that all possible stake-holders have been involved in the requirements acquisition process, and that possible conflicts and other inconsistencies have been obviated.

Rule 4 Requirements Implementability: Make sure that requirements are implementable. That is, do not express a requirement for which you have no assurance that it can be implemented. In other words, although the requirements phase is not a design phase, one must tacitly assume, perhaps even indicate, somehow, that an implementation is possible. But the requirements in and by themselves, may stay short of expressing such designs. Caveat: The domain and requirements specifications are, in our approach, model-oriented. That helps expressing ‘implementability’.

Definition: 71 Requirements (II): By requirements we shall understand a document which prescribes desired properties of a machine: what endurants the machine shall “maintain”, and what the machine shall (must; not should) offer of functions and of behaviours while also expressing which events the machine shall “handle”.

9 ■ marks the end of a rule.
By a machine that “maintains” endurants we shall mean: a machine which, “between” users’ use of that machine, “keeps” the data that represents these entities. From earlier we repeat:

**Definition:** 72 Machine: By machine we shall understand a, or the, combination of hardware and software that is the target for, or result of the required computing systems development.

So this, then, is a main objective of requirements development: to start towards the design of the hardware + software for the computing system.

**Definition:** 73 Requirements (III): To specify the machine.

When we express requirements and wish to “convert” such requirements to a realisation, i.e., an implementation, then we find that some requirements (parts) imply certain properties to hold of the hardware on which the software to be developed is to “run”, and, obviously, that remaining — probably the larger parts of the — requirements imply certain properties to hold of that software.

... 

Whereas domain descriptions may describe phenomena that cannot be computed, requirements prescriptions must describe computable phenomena.

### 8.3.1 Some Requirements Aspects

We shall unravel requirements in two stages — (i) the first stage is sketchy (and thus informal) (ii) while the last stage is systematic and both informal and formal. The sketchy stage consists of (i.1) a narrative problem/objective sketch, (i.2) a narrative system requirements sketch, and (i.3) a narrative user & external equipment requirements sketch. (ii) The narrative and formal stage consists of design assumptions and design requirements. It is systematic, and mandates both strict narrative and formal prescriptions. And it is “derivable” from the domain description. In a sense stage (i) is made superfluous once stage (ii) has been completed. The formal, engineering design work is to based on stage (ii). The purpose of the two stages (i–ii) is twofold: to gently lead the requirements engineer and the reader into the requirements problems while leading the requirements engineer and reader to focus on the very requirements essentials.

### 8.3.1.1 Requirements Sketches

#### 8.3.1.1.1 Problem, Solution and Objective Sketch

**Definition:** 74 Problem, Solution and Objective Sketch: By a problem, solution and objective sketch we understand a narrative which emphasises what the problem to be solved is, outlines a possible solution and sketches an objective of the solution.

**Example 109** The problem is that of traffic congestion. The chosen solution is to [build and] operate a toll-road system integrated into a road net and charge toll-road users a usage fee. The objective is therefore to create a road-pricing product. By a road-pricing product we shall understand an information technology-based system containing computers and communications equipment and software that enables the recording of vehicle movements within the toll-road and thus enables the owner of the road net to charge the owner of the vehicles fees for the usage of that toll-road.
8.3.1.2 Systems Requirements

Definition: 75 System Requirements: By a system requirements narrative we understand a narrative which emphasises the overall assumed and/or required hardware and software system equipment.

Example 110 The requirements are based on the following constellation of system equipment: (i) there is assumed a GNSS: a Global Navigation Satellite System; (ii) there are vehicles equipped with GNSS receivers; (iii) there is a well-delineated road net called a toll-road net with specially equipped toll-gates with vehicle identification sensors, exit barriers which afford (only specially equipped) vehicles to exit from the toll-road net; and (iv) there is a road-pricing calculator.

The system to be designed (from the requirements) is the road-pricing calculator. These four system elements are required to behave and interact as follows: (a) The GNSS is assumed to continuously offer vehicles information about their global position; (b) vehicles shall contain a GNSS receiver which based on the global position information shall regularly calculate their timed local position and offer this to the calculator — while otherwise cruising the general road net as well as the toll-road net, the latter while carefully moving through toll-gates; (c) toll-gates shall register the identity of vehicles passing the toll-road and offer this information to the calculator; and (d) the calculator shall accept all messages from vehicles and gates and use this information to record the movements of vehicles and bill these whenever they exit the toll-road. The requirements are therefore to include assumptions about [1] the GNSS satellite and telecommunications equipment, [2] the vehicle GNSS receiver equipment, [3] the vehicle handling of GNSS input and forwarding, to the road pricing system, of its interpretation of GNSS input, [4] the toll-gate sensor equipment, [5] the toll-gate barrier equipment, [6] the toll-gate handling of entry, vehicle identification and exit sensors and the forwarding of vehicle identification to the road pricing calculator, and [7] the communications between toll-gates and vehicles, on “one side”, and the road pricing calculator, on the “other side”. It is in this sense that the requirements are for an information technology-based system of both software and hardware — not just hard computer and communications equipment, but also movement sensors and electro-mechanical “gear”.

8.3.1.3 User and External Equipment Requirements

Definition: 76 User and External Equipment Requirements: By a user and external equipment requirements narrative we understand a narrative which emphasises assumptions about the human user and external equipment interfaces to the system components.

The user and external equipment requirements detail, and thus make explicit, the assumptions listed in Example 110.

Example 111 The human users of the road-pricing system are: (a) vehicle drivers, (b) toll-gate sensor, actuator and barrier service staff, and (c) the road-pricing calculator service staff. The external equipment are: (1) firstly, the GNSS satellites and the telecommunications equipment which enables communication between (i) the GNSS satellites and vehicles, (ii) vehicles and the road-pricing calculator and (iii) toll-gates and the road-pricing calculator. Moreover, the external equipment are (2) the toll-gates with their sensors: entry, vehicle identity, and exit, and the barrier actuator. The external equipment are, finally, (3), the vehicles.

That is, although we do indeed exemplify domain and requirements aspects of users and external equipment, we do not expect to machine, i.e., to hardware or software design these elements; they are assumed already implemented.
8.3 Requirements

8.3.1 The Narrative and Formal Requirements Stage

8.3.1.2 Assumption and Design Requirements

Definition: 77 Assumption and Design Requirements: By *assumption and design requirements* we understand precise prescriptions of the endurants and perdurants of the (to be designed) system components and the assumptions which that design must rely upon.■

The specification principles, techniques and tools of expressing design and assumptions, upon which the design can be relied, will be covered and exemplified, extensively, in Sects. 8.4–8.5.

8.3.2 The Three Phases of Requirements Engineering

There are, as we see it, three kinds of design assumptions and requirements: (i) domain requirements, (ii) interface requirements and (iii) machine requirements. (i) *Domain requirements* are those requirements which can be expressed solely using terms of the domain ■ (ii) *Interface requirements* are those requirements which can be expressed only using technical terms of both the domain and the machine ■ (iii) *Machine requirements* are those requirements which, in principle, can be expressed solely using terms of the machine ■

Definition: 78 Verification Paradigm: Some preliminary designations: let $D$ designate the the domain description; let $R$ designate the requirements prescription, and let $S$ designate the system design. Now $D, S \models R$ shall be read: it must be verified that the *system design* satisfies the *requirements prescription* in the context of the *domain description* ■

The “in the context of $D$...” term means that proofs of *software design correctness* with respect to requirements will often have to refer to *domain requirements assumptions*. We refer to [185, Gunter, Jackson and Zave, 2000] for an analysis of a varieties of forms in which $\models$ relate to variants of $D$, $R$ and $S$.

8.3.3 Order of Presentation of Requirements Prescriptions

The *domain requirements development* stage — as we shall see — can be sub-staged into: projection, instantiation, determination, extension and fitting. The *interface requirements development* stage — can be sub-staged into shared: endurant, action, event and behaviour developments, where “sharedness” pertains to phenomena shared between, i.e., “present” in, both the domain (concretely, manifestly) and the machine (abstractly, conceptually). These development stages need not be pursued in the order of the three stages and their sub-stages. We emphasize that one thing is the stages and steps of development, as for example these: projection, instantiation, determination, extension, fitting, shared endurants, shared actions, shared events, shared behaviours, et cetera, another thing is the requirements prescription that results from these development stages and steps. The further software development, after and on the basis of the requirements prescription starts only when all stages and steps of the requirements prescription have been fully developed. The domain engineer is now free to rearrange the final prescription, irrespective of the order in which the various sections were developed, in such a way as to give a most pleasing, pedagogic and cohesive reading (i.e., presentation). From such a requirements prescription one can therefore not necessarily see in which order the various sections of the prescription were developed.

8.3.4 Design Requirements and Design Assumptions

A crucial distinction is between *design requirements* and *design assumptions*. The *design requirements* are those requirements for which the system designer has to implement hardware or software in order satisfy system user expectations ■ The *design assumptions* are those requirements for which the
system designer does not have to implement hardware or software, but whose properties the designed hardware, respectively software relies on for proper functioning.

### Requirements: Road Pricing, Design Requirements

**Example 112** The design requirements for the road pricing calculator of this chapter are for the design of (i) that part of the vehicle software which interfaces the GNSS receiver and the road pricing calculator (cf. Items 414–417), (ii) of that part of the toll-gate software which interfaces the toll-gate and the road pricing calculator (cf. Items 422–424) and (i) of the road pricing calculator (cf. Items 453–466).

### Requirements: Road Pricing, Design Assumptions

**Example 113** The design assumptions for the road pricing calculator include: (i) that vehicles behave as prescribed in Items 413–417, (ii) that the GNSS regularly offers vehicles correct information as to their global position (cf. Item 414), (iii) that toll-gates behave as prescribed in Items 419–424, and (iv) that the road net is formed and well-formed as defined in Examples 118 – 120.

### Requirements: Road Pricing, Toll-Gate System, Design Requirements

**Example 114** The design requirements for the toll-gate system of this chapter are for the design of software for the toll-gate and its interfaces to the road pricing system, i.e., Items 418–419.

### Requirements: Road Pricing, Toll-Gate System, Design Assumptions

**Example 115** The design assumptions for the toll-gate system include (i) that the vehicles behave as per Items 413–417, and (ii) that the road pricing calculator behave as per Items 453–466.

### 8.3.5 Derived Requirements

In building up the domain, interface and machine requirements, a number of machine concepts are introduced. These machine concepts enable the expression of additional requirements. It is these we refer to as derived requirements. Techniques and tools espoused in such classical publications as [134, 253, 380, 264, 355] can in those cases be used to advantage.

### 8.4 Domain Requirements

Domain requirements primarily express the assumptions that a design must rely upon in order that that design can be verified. Although domain requirements firstly express assumptions it appears that the software designer is well-advised in also implementing, as data structures and procedures, the endurants, respectively perdurants expressed in the domain requirements prescriptions. Whereas domain endurants are “real-life” phenomena they are now, in domain requirements prescriptions, abstract concepts (to be represented by a machine).

**Definition:** Domain Requirements Prescription: A domain requirements prescription is that subset of the requirements prescription whose technical terms are defined in a domain description.

To determine a relevant subset all we need is collaboration with requirements, cum domain stake-holders. Experimental evidence, in the form of example developments of requirements prescriptions from domain descriptions, appears to show that one can formulate techniques for such developments around a few domain-description-to-requirements-prescription operations. We suggest these: projection, instantiation, determination, extension and fitting. In Sect. 8.3.3 we mentioned that the order in which one performs these...
8.4 Domain Requirements

Domain Requirements-prescription operations is not necessarily the order in which we have listed them here, but, with notable exceptions, one is well-served in starting out requirements development by following this order.

8.4.1 Domain Projection

Definition: 80 Domain Projection: By a domain projection is meant a subset of the domain description, one which projects out all those endurants: parts, materials and components, as well as perdurants: actions, events and behaviours that the stake-holders do not wish represented or relied upon by the machine.

The resulting document is a partial domain requirements prescription. In determining an appropriate subset the requirements engineer must secure that the final “projection prescription” is complete and consistent — that is, that there are no “dangling references”, i.e., that all entities and their internal properties that are referred to are all properly defined.

8.4.1.1 Domain Projection — Narrative

We now start on a series of examples that illustrate domain requirements development.

Example 116 We require that the road pricing system shall [at most] relate to the following domain entities — and only to these: the net, its links and hubs, and their properties (unique identifiers, mereologies and some attributes), the vehicles, as endurants, and the general vehicle behaviours, as perdurants. We treat projection together with a concept of simplification. The example simplifications are vehicle positions and, related to the simpler vehicle position, vehicle behaviours. To prescribe and formalise this we copy the domain description. From that domain description we remove all mention of the hub insertion action, the link disappearance event, and the monitor.

As a result we obtain \( \Delta \), the projected version of the domain requirements prescription\(^{12}\).

8.4.1.2 Domain Projection — Formalisation

The requirements prescription hinges, crucially, not only on a systematic narrative of all the projected, instantiated, determined, extended and fitted specifications, but also on their formalisation. In the formal domain projection example we, regretfully, omit the narrative texts. In bringing the formal texts we keep the item numbering from Sect. 8.2, where you can find the associated narrative texts.

Example 117 Main Sorts

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Domain Projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>286 ( \Delta )</td>
<td>286a N( \rho )</td>
<td></td>
</tr>
<tr>
<td>286b F( \rho )</td>
<td>286a obs( N_{\rho} \rightarrow \Delta_{\rho} \rightarrow N_{\rho} )</td>
<td></td>
</tr>
<tr>
<td>286b obs( F_{\rho} \rightarrow \Delta_{\rho} \rightarrow F_{\rho} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Concrete Types

12 Restrictions of the net to the toll road nets, hinted at earlier, will follow in the next domain requirements steps.
For ‘road pricing’ we need vehicle positions. But, for “technical reasons”, we must abstain from using vehicle positions. We therefore simplify vehicle position designates to either a link or a hub.

**Attributes:** We project attributes of hubs, links and vehicles.

**Mereology**

**Unique Identifiers**

**Type**

- 292a hubs: $H_{\delta}$ → $H_{\delta}$
- 292c uid: $V_{\delta}$ → $V_{\delta}$

**Value**

- 297 $\text{obs}_{\text{meero}}$: $H_{\delta}$ → $L_{\delta}$-set
- 298 $\text{obs}_{\text{meero}}$: $L_{\delta}$ → $H_{\delta}$-set
- 299 $\text{obs}_{\text{meero}}$: $V_{\delta}$ → $V_{\delta}$-set

**Axiom**

- 301 $\forall \delta\cdot \Delta_{\delta}$, $\text{hs}: H_{\delta}$ → $H_{\delta}$
- 302 $\forall f: F_{\delta}$ → $V_{\delta}$

**Assume**

- 303 hubs: $H_{\delta}$ → $H_{\delta}$
- 304 $\text{attrs}_{\delta}$: $H_{\delta}$ → $H_{\delta}$
- 305 $\text{attrs}_{\delta}$: $L_{\delta}$ → $L_{\delta}$

Finally **vehicles:** For ‘road pricing’ we need vehicle positions. But, for “technical reasons”, we must abstain from using vehicle positions. We therefore simplify vehicle positions.

**Type**

- 336 $\text{SVPos} = \text{SonL} | \text{SatH}$
8.4 Domain Requirements

Domain Requirements

\[ \forall n: N, \text{SonL}(li): \text{SVPos} \]
\[ \exists l: L \in \text{obs}\_L(\text{obs}\_N(n)) \Rightarrow li = \text{uid}\_L(l) \]

[Global Values]

\[ \forall n: N, \text{SatH}(hi): \text{SVPos} \]
\[ \exists h: H \in \text{obs}\_H(\text{obs}\_N(n)) \Rightarrow hi = \text{uid}\_H(h) \]

[Behaviour Signatures]

We omit the monitor behaviour.

We leave the vehicle behaviours’ attribute argument undefined.

[The System Behaviour]

We omit the monitor behaviour.

[The Vehicle Behaviour]

Given the simplification of vehicle positions we simplify the vehicle behaviour given in Items 331–332.

We can simplify Items 331’–332c’ further.

This line coalesces Items 331’ and 332’.

Coalescing Items 331a’ and 332a’.

Captures the distinct parameters of Items 331’ and 332’.

Item 331(b)i’.

July 6, 2020, 10:05, A Foundation for Software Development © Dines Bjørner 2020, Fredsvej 11, DK–2840 Holte, Denmark
8.4.1.3 Discussion

Domain projection can also be achieved by developing a “completely new” domain description — typically on the basis of one or more existing domain description(s) — where that “new” description now takes the role of being the project domain requirements.

8.4.2 Domain Instantiation

**Definition:** Domain Instantiation: By domain instantiation we mean a refinement of the partial domain requirements prescription (resulting from the projection step) in which the refinements aim at rendering more concrete, more specific the endurants: parts and materials, as well as the perdurants: actions, events and behaviours of the domain requirements prescription. Instantiations usually render these concepts less general.

Properties that hold of the projected domain shall also hold of the (therefrom) instantiated domain.

Refinement of endurants can be expressed (i) either in the form of concrete types, (ii) or of further “delineating” axioms over sorts, (iii) or of a combination of concretisation and axioms. We shall exemplify the third possibility. Example 118 expresses requirements that the road net (on which the road-pricing system is to be based) must satisfy. Refinement of perdurants will not be illustrated (other than the simplification of the vehicle projected behaviour).

8.4.2.1 Domain Instantiation

**Requirements: Domain Requirements, Instantiation – Road Net 1/2**

**Example 118** We now require that there is, as before, a road net, $n_{r}:N_{r}$, which can be understood as consisting of two, “connected sub-nets”. A toll-road net, $trn_{r}:TRN_{r}$, cf. Fig. 8.1 on the facing page, and an ordinary road net, $n_{o}$. The two are connected as follows: The toll-road net, $trn_{r}$, borders some toll-road plazas, in Fig. 8.1 on the next page shown by white filled circles (i.e., hubs). These toll-road plaza hubs are proper hubs of the ‘ordinary’ road net, $n_{o}$.

The instantiated domain, $\delta_{r}:\Delta_{r}$, has just the net, $n_{r}:N_{r}$ being instantiated.

The road net consists of two “sub-nets”

- an “ordinary” road net, $n_{o}:N_{o}$, and
- a toll-road net proper, $trn:TRN_{r}$ —

  c “connected” by an interface hil:HIL:

  i That interface consists of a number of toll-road plazas (i.e., hubs), modeled as a list of hub identifiers, hil:HIL.

  ii The toll-road plaza interface to the toll-road net, $trn:TRN_{r}$, has each plaza, hil[i], connected to a pair of toll-road links: an entry and an exit link: $(l_{e},l_{x})$.

  iii The toll-road plaza interface to the ‘ordinary’ net, $n_{o}:N_{o}$, has each plaza, i.e., the hub designated by the hub identifier hil[i], connected to one or more ordinary net links, $\{l_{1},l_{2},\ldots,l_{k}\}$.

The three lists have commensurate lengths ($\ell$).
8.4 Domain Requirements

Example 118 (Continued) \( \ell \) is the number of toll plazas, hence also the number of toll-road intersection hubs and therefore a number one larger than the number of pairs of main toll-road ("up" and "down") links

\[
\Delta_{\mathcal{F}} = N_{\mathcal{F}} \times \text{HIL} \times \text{TRN}
\]

\[
N_{\mathcal{F}} \times \text{HIL} \times \text{TRN}
\]

\[
\text{TRN}_{\mathcal{F}} = (L \times L)^* \times H^* \times (L \times L)^*
\]

We have named the “ordinary" net sort (primed) \( N_{\mathcal{F}}' \). It is “almost" like (unprimed) \( N_{\mathcal{F}} \) — except that the interface hubs are also connected to the toll-road net entry and exit links. The partial concretisation of the net sorts, \( N_{\mathcal{F}}' \), into \( N_{\mathcal{I}} \) requires some additional well-formedness conditions to be satisfied.

348 The toll-road intersection hubs all\(^{15} \) have distinct identifiers.

348 \( \text{wf-toll-road-intersect-hub-ids: H}^* \rightarrow \text{Bool} \)

348 \( \text{wf-toll-road-intersect-hub-ids(hl)} \equiv \text{len hl} = \text{card xtr_his(hl)} \)

349 The toll-road links all have distinct identifiers.

349 \( \text{wf-toll-road-up-down-link-ids: (L \times L)^*} \rightarrow \text{Bool} \)

349 \( \text{wf-toll-road-up-down-link-ids(lll)} \equiv \text{2 \times len lll} = \text{card xtr_lis(lll)} \)

350 The toll-road entry/exit links all have distinct identifiers.

350 \( \text{wf-ex-link-ids: (L \times L)^*} \rightarrow \text{Bool} \)

350 \( \text{wf-ex-link-ids(exll)} \equiv \text{2 \times len exll} = \text{card xtr_lis(exll)} \)

351 Proper net links must not designate toll-road intersection hubs.

351 \( \text{wf-isotd-toll-road-intersect-ids: H}^* \times H^* \rightarrow N_{\mathcal{F}}' \rightarrow \text{Bool} \)

351 \( \text{wf-isotd-toll-road-intersect-ids(hil,hl)(n_{\mathcal{F}})} \equiv \)

351 \( \text{let ls=xtr_lis(n_{\mathcal{F}}) in} \)

351 \( \text{let his = \{ mereo_L(l) | L \subseteq ls \} in} \)

351 \( \text{his \cap xtr_his(hl) = \{} \text{ end end} \)
The plaza hub identifiers must designate hubs of the ‘ordinary’ net.

\[ \text{wf}_p \text{hubs} \rightarrow \text{ord}_p \text{ord}_n \text{ord}_n \text{net} : \text{H}' \rightarrow N'_A \rightarrow \text{Bool} \]

\[ \text{wf}_p \text{hubs} \rightarrow \text{ord}_p \text{ord}_n \text{ord}_n \text{net}(h_i(n'_A)) \equiv \text{elems \ hil} \subseteq \text{xstrl}(n'_A) \]

The plaza hub mereologies must each identify at least one hub of the ordinary net, a besides identifying the two entry/exit links with which they are supposed to be connected.

\[ \text{wf}_p \text{hubs} \rightarrow \text{ord}_p \text{ord}_n \text{ord}_n \text{net} : N'_A \rightarrow \text{Bool} \]

\[ \forall i : \text{Nat} \cdot i \in \text{inds \ exll} \Rightarrow \text{let} \ h = \text{getH}(\text{hil}(i))(n'_A) \ in \]

The mereology of each toll-road intersection hub must identify exactly the entry/exit links c with which they are supposed to be connected.

\[ \text{wf}_t \text{oll}_r \text{oad \ jsect}_h \text{ub \ iface} : N_F \rightarrow \text{Bool} \]

\[ \forall i : \text{Nat} \cdot i \in \text{inds \ exll} \Rightarrow \text{let} \ h = \text{getH}(\text{hil}(i))(n'_A) \ in \]

The mereology of the entry/exit links must identify a the entry/exit links hub toll-road intersection hubs b with which they are supposed to be connected.

\[ \text{wf}\_\text{exll} : (L \times L)^* \times \text{H}' \times \text{H}' \rightarrow \text{Bool} \]

\[ \forall i : \text{Nat} \cdot i \in \text{len \ exll} \Rightarrow \text{let} \ (h_i, \text{exll}(i), h_i) = (\text{hil}(i), \text{exll}(i), \text{hil}(i)) \ in \]

We have used some additional auxiliary functions:

\[ \text{xstrl}(t_1, t_2) \equiv \{ \text{uidH}(1) \cup \{ \text{uidH}(1) \} \} \]

\[ \text{xstrl}(L \times L)^* \rightarrow \text{L-Set} \]

\[ \text{xstrl}(t_1, t_2) \subseteq \{ \text{uidH}(1) \cup \{ \text{uidH}(1) \} \} \]

\[ \text{uidH}(h_i) \in \text{elems \ hil} \]

The well-formedness of instantiated nets is now the conjunction of the individual well-formedness predicates above.

\[ \text{wf}\_\text{instantiated} : N_F \rightarrow \text{Bool} \]

\[ \text{wf}\_\text{instantiated} : \text{net}(n'_A, \text{hil}, (\text{exll}, \text{hil}, \text{III})) \]

\[ \text{wf} \rightarrow \text{dist}_t \text{oll}_r \text{oad \ jsect}_h \text{ub \ jds}(\text{hil}) \]

\[ \text{wf} \rightarrow \text{dist}_t \text{oll}_r \text{oad \ jsect}_h \text{ub \ jds}(\text{hil}) \]

\[ \text{wf} \rightarrow \text{dist}_t = \text{dist}_o \text{link \ jds}(\text{exll}) \]

\[ \text{wf} = \text{dist}_t = \text{dist}_o \text{link \ jds}(\text{exll}) \]

\[ \text{wf} = \text{dist}_t = \text{dist}_o \text{link \ jds}(\text{exll}) \]

\[ \text{wf} = \text{dist}_t = \text{dist}_o \text{link \ jds}(\text{exll}) \]

\[ \text{wf} = \text{dist}_t = \text{dist}_o \text{link \ jds}(\text{exll}) \]

\[ \text{wf} = \text{dist}_t = \text{dist}_o \text{link \ jds}(\text{exll}) \]
8.4 Domain Requirements

8.4.2 Domain Instantiation — Abstraction

Requirements: Domain Requirements, Instantiation of Road Net, Abstraction

Example 119 Domain instantiation has refined an abstract definition of net sorts, \( n_{\mathcal{P}} : N_{\mathcal{P}} \), into a partially concrete definition of nets, \( n_{\mathcal{I}} : N_{\mathcal{I}} \). We need to show the refinement relation:

- \( \text{abstraction}(n_{\mathcal{I}}) = n_{\mathcal{P}} \).

\[
\begin{align*}
\text{value} & \quad \text{abstraction}: N_{\mathcal{I}} \to N_{\mathcal{P}} \\
\text{abstraction}(n_{\mathcal{I}}) & \equiv \text{let } n_{\mathcal{P}} : N_{\mathcal{P}} \cdot \\
\text{hs} & = \text{obs(HS}_{\mathcal{P}}(\text{obs(HA}_{\mathcal{P}}(n_{\mathcal{I}}))), \\
\text{ls} & = \text{obs(LS}_{\mathcal{P}}(\text{obs(LA}_{\mathcal{P}}(n_{\mathcal{I}}))), \\
\text{ths} & = \text{elems hl}, \\
\text{eells} & = \text{xtr(links(eell)}, \text{llls} = \text{xtr(links(lll)}) \text{ in} \\
\text{hs} & \cup \text{ths} = \text{obs(HS}_{\mathcal{P}}(\text{obs(HA}_{\mathcal{P}}(n_{\mathcal{P}}))), \\
\text{ls} & \cup \text{eells} \cup \text{llls} = \text{obs(LS}_{\mathcal{P}}(\text{obs(LA}_{\mathcal{P}}(n_{\mathcal{P}})))) \\
\text{n}_{\mathcal{P}} & \text{end end}
\end{align*}
\]

358 The abstraction function takes a concrete net, \( n_{\mathcal{I}} : N_{\mathcal{I}} \), and yields an abstract net, \( n_{\mathcal{P}} : N_{\mathcal{P}} \).

359 The abstraction function doubly decomposes its argument into constituent lists and sub-lists.

360 There is postulated an abstract net, \( n_{\mathcal{P}} : N_{\mathcal{P}} \), such that

361 the hubs of the concrete net and toll-road equals those of the abstract net, and

362 the links of the concrete net and toll-road equals those of the abstract net.

363 And that abstract net, \( n_{\mathcal{P}} : N_{\mathcal{P}} \), is postulated to be an abstraction of the concrete net.

8.4.2.3 Discussion

Domain descriptions, such as illustrated in [70, Manifest Domains: Analysis & Description] and in this chapter, model families of concrete, i.e., specifically occurring domains. Domain instantiation, as exemplified in this section (i.e., Sect. 8.4.2), “narrow down” these families. Domain instantiation, such as it is defined, cf. Definition 81 on Page 216, allows the requirements engineer to instantiate to a concrete instance of a very specific domain, that, for example, of the toll-road between Bolzano Nord and Trento Sud in Italy (i.e., \( n=7 \))\(^{16}\).

8.4.3 Domain Determination

Definition: 82 Determination: By domain determination we mean a refinement of the partial domain requirements prescription, resulting from the instantiation step, in which the refinements aim at rendering less non-determinate, more determinate the endurants: parts and materials, as well as the perdurants: functions, events and behaviours of the partial domain requirements prescription.

Determinations usually render these concepts less general. That is, the value space of endurants that are made more determinate is “smaller”, contains fewer values, as compared to the endurants before determination has been applied”.

\(^{16}\) Here we disregard the fact that this toll-road does not start/end in neither Bolzano Nord nor Trento Sud.
8.4.3.1 Domain Determination: Example

We show an example of ‘domain determination’. It is expressed solely in terms of axioms over the concrete toll-road net type.

Example 120  We focus only on the toll-road net. We single out only two ‘determinations’: All Toll-road Links are One-way Links:

The entry/exit and toll-road links
a) are always all one way links,

b) as indicated by the arrows of Fig. 8.1 on Page 217,

c) such that each pair allows traffic in opposite directions.

Predicates 364a′ and 364a″ express the same property.

All Toll-road Hubs are Free-flow

The hub state spaces are singleton sets of the toll-road hub states which always allow exactly these (and only these) crossings:

a) from entry links back to the paired exit links,

b) from entry links to emanating toll-road links,

c) from incident toll-road links to exit links, and

d) from incident toll-road link to emanating toll-road links.

Predicates 365a: from entry links back to the paired exit links:

Predicates 365b: from entry links to emanating toll-road links:

Predicates 365c: from incident toll-road links to exit links:

Predicates 365d: from incident toll-road link to emanating toll-road links:
The em and in in the toll-road link list \((\text{em} \times \text{in})^*\) designate selectors for emanating, respectively incident links. 365c: from incident toll-road links to exit links:

\[
365c \quad \sigma \text{txls}: (L \times (\text{em} \times \text{in}))^* \rightarrow L^\Sigma
\]

case i of

\[
365c \quad \text{len} \quad \text{ll} + 1 \rightarrow \{(\text{id}_{\text{LL}}(\text{in}(\text{ll}(i))), \text{id}_{\text{LL}}(\text{em}(\text{ll}(i))))\},
\]

\[
365c \quad \text{len} \quad \text{ll} - 1 \rightarrow \{(\text{id}_{\text{LL}}(\text{in}(\text{ll}(i-1))), \text{id}_{\text{LL}}(\text{em}(\text{ll}(i))))\},
\]

\[
365c \quad \text{len} \quad \text{ll} \quad \text{ll} \quad \rightarrow \{(\text{id}_{\text{LL}}(\text{in}(\text{ll}(i))), \text{id}_{\text{LL}}(\text{em}(\text{ll}(i))))\}
\]

end

365d: from incident toll-road link to emanating toll-road links:

\[
365d \quad \sigma \text{ttls}: \text{Nat} \times (\text{em} \times \text{in})^* \rightarrow L^\Sigma
\]

case i of

\[
365d \quad \text{len} \quad \text{ll} \quad \rightarrow \{(\text{id}_{\text{LL}}(\text{in}(\text{ll}(i))), \text{id}_{\text{LL}}(\text{em}(\text{ll}(i))))\},
\]

\[
365d \quad \text{len} \quad \text{ll} \quad \rightarrow \{(\text{id}_{\text{LL}}(\text{in}(\text{ll}(i-1))), \text{id}_{\text{LL}}(\text{em}(\text{ll}(i))))\},
\]

\[
365d \quad \text{len} \quad \text{ll} \quad \rightarrow \{(\text{id}_{\text{LL}}(\text{in}(\text{ll}(i))), \text{id}_{\text{LL}}(\text{em}(\text{ll}(i))))\}
\]

end

The example above illustrated ‘domain determination’ with respect to endurants. Typically “endurant determination” is expressed in terms of axioms that limit state spaces — where “endurant instantiation” typically “limited” the mereology of endurants: how parts are related to one another. We shall not exemplify domain determination with respect to perdurants.

### 8.4.3.2 Discussion

The borderline between instantiation and determination is fuzzy. Whether, as an example, fixing the number of toll-road intersection hubs to a constant value, e.g., \(n=7\), is instantiation or determination, is really a matter of choice!

### 8.4.4 Domain Extension

**Definition:** 83 **Extension:** By domain extension we understand the introduction of endurants (see Sect. 8.4.4.1) and perdurants (see Sect. 8.5.2) that were not feasible in the original domain, but for which, with computing and communication, and with new, emerging technologies, for example, sensors, actuators and satellites, there is the possibility of feasible implementations, hence the requirements, that what is introduced becomes part of the unfolding requirements prescription.

### 8.4.4.1 Endurant Extensions

**Definition:** 84 **Endurant Extension:** By an endurant extension we understand the introduction of one or more endurants into the projected, instantiated and determined domain \(D_R\) resulting in domain \(D_R'\) such that these form a conservative extension of the theory, \(T_{D_R}\) denoted by the domain requirements \(D_R\) (i.e., “before” the extension), that is: every theorem of \(T_{D_R}\) is still a theorem of \(T_{D_R'}\).

Usually domain extensions involve one or more of the already introduced sorts. In Example 121 on the following page we introduce (i.e., “extend”) vehicles with GPSS-like sensors, and introduce toll-gates with entry sensors, vehicle identification sensors, gate actuators and exit sensors. Finally road pricing calculators are introduced.

**Requirements:** Domain Requirements, Endurant Extension 1/2
Example 121  We present the extensions in several steps. Some of them will be developed in this section. Development of the remaining will be deferred to Sect. 8.5.1.3. The reason for this deferment is that those last steps are examples of interface requirements. The initial extension-development steps are: [a] vehicle extension, [b] sort and unique identifiers of road price calculators, [c] vehicle to road pricing calculator channel, [d] sorts and dynamic attributes of toll-gates, [e] road pricing calculator attributes, [f] “total” system state, and [g] the overall system behaviour. This decomposition establishes system interfaces in “small, easy steps”.

8.4.4.1.1  [a] Vehicle Extension:

366 There is a domain, \( \delta_E:\Delta_E \), which contains a fleet, \( f:F_E \), that is,
368 a set, \( vs:V_E \), of extended vehicles, \( v:V_E \) — their extension amounting to a dynamic reactive attribute, whose value, \( ti\text{-}gpos:TiGpos \), at any time, reflects that vehicle’s time-stamped global position.\(^{17}\)

371 The vehicle’s GNSS receiver calculates, \( loc\text{-}pos \), its local position, \( lpos:LPos \), based on these signals. Vehicles access these external attributes via the external attribute channel, \( attr\text{-}TiGPos\text{-}ch \).

377 To access these external attributes, a fleet, \( f: obs\text{-}F_E(\delta_E) \), observes its set of vehicles, \( obs\text{-}VS_E : F_E \rightarrow VS_E \), which, in turn, observes their unique identifiers, \( obs\text{-}VI(v) \) via the external attribute channel, \( attr\text{-}TiGPos\text{-}ch \).

We define two auxiliary functions,

\begin{align*}
\text{value} & : \delta_E : \Delta_E \rightarrow F_E,
\text{channel} & : \delta_E : \Delta_E \rightarrow \text{set},
\end{align*}

\[ xtr\text{-}vs: (\Delta_E | F_E | VS_E) \rightarrow V_E \]

\[ xtr\text{-}vs(arg) \equiv \{ uid\text{-}VI(v) \mid v \in xtr\text{-}vs(arg) \} \]

8.4.4.1.2  [b] Road Pricing Calculator: Basic Sort and Unique Identifier:

375 The domain \( \delta_E:\Delta_E \), also contains a pricing calculator, \( c:C \), with unique identifier \( ci:CI \).

\[ \text{type} \quad c:C, \text{ci:CI} \]

8.4.4.1.3  [c] Vehicle to Road Pricing Calculator Channel:

376 Vehicles can, on their own volition, offer the timed local position, \( vit\text{-}lpos:VITilPos \), to the pricing calculator, \( c:C \), along a vehicles-to-calculator channel, \( v\text{-}c\text{-}ch \).

\[ \text{type} \quad \text{VITilPos} = V \times (T \times LPos) \]

\[ \text{channel} \]
8.4 Domain Requirements

8.4.4.1.4 [d] Toll-gate Sorts and Dynamic Types:

We extend the domain with toll-gates for vehicles entering and exiting the toll-road entry and exit links. Figure 8.2 illustrates the idea of gates.

![Toll Gate Diagram](image)

**Fig. 8.2. Toll Gate**

**Requirements: Domain Requirements, Endurant Extension 2/2**

**Example 121 (Continued)** Figure 8.2 is intended to illustrate a vehicle entering (or exiting) a toll-road arrival link. The toll-gate is equipped with three sensors: an arrival sensor, a vehicle identification sensor and an departure sensor. The arrival sensor serves to prepare the vehicle identification sensor. The departure sensor serves to prepare the gate for closing when a vehicle has passed. The vehicle identification sensor identifies the vehicle and “delivers” a pair: the current time and the vehicle identifier. Once the vehicle identification sensor has identified a vehicle the gate opens and a message is sent to the road pricing calculator as to the passing vehicle’s identity and the identity of the link associated with the toll-gate (see Items 394-395 on the next page).

The domain contains the extended net, \( n: N_E \), with the net extension amounting to the toll-road net, \( TRN_E \), that is, the instantiated toll-road net, \( trn:TRN_E \), is extended, into \( trn:TRN_E \), with entry, eg:EG, and exit, xg:XG, toll-gates.

From entry- and exit-gates we can observe

- their unique identifier and
- their mereology: pairs of entry-, respectively exit link and calculator unique identifiers; further
- a pair of gate entry and exit sensors modeled as *external attribute* channels, (ges:ES,gls:XS), and
- a time-stamped vehicle identity sensor modeled as *external attribute* channels.

**type**

- \( N_E \)
- \( TRN_E = (EG \times XG)^* \times TRN_E \)
- \( GI \)

**value**

- \( obs_{N_E}: \Delta_E \rightarrow N_E \)
- \( obs_{TRN_E}: N_E \rightarrow TRN_E \)
- \( uid_{GI}: (EG(XG) \rightarrow GI \)
- \( mereo_{GI}: (EG(XG) \rightarrow (LI \times CI) \)
- \( trn:TRN_E = obs_{TRN_E}(\delta_E) \)

**channel**

- \( \{ attr_{entry}.ch[gi]|gi:GI|xr.xGids(trn) \} "enter" \)
- \( \{ attr_{exit}.ch[gi]|gi:GI|xr.xGids(trn) \} "exit" \)
We define some auxiliary functions over toll-road nets, \( \text{trn} : \text{TRN} \rightarrow \cdot \):

- \( \text{xtr}_{eG} : \text{TRN} \rightarrow \text{EG} \)
  - extracts the list of entry gates,
- \( \text{xtr}_{xG} : \text{TRN} \rightarrow \text{XG} \)
  - extracts the list of exit gates,
- \( \text{xtr}_{Gids} : \text{TRN} \rightarrow \text{GI} \)
  - extracts the set of entry gate identifiers,
- \( \text{xtr}_{xGIds} : \text{TRN} \rightarrow \text{GI} \)
  - extracts the set of exit gate identifiers,
- \( \text{xtr}_{Gs} : \text{TRN} \rightarrow \text{G} \)
  - extracts the set of all gates, and
- \( \text{xtr}_{GIds} : \text{TRN} \rightarrow \text{GI} \)
  - extracts the set of all gate identifiers.

A well-formedness condition expresses

\[ \forall n : N \cdot \text{trn} : \text{TRN} \rightarrow \cdot \]

\[ \text{let} \ (\text{exgl},(\text{exl},\text{hl},\text{lll})) = \text{obs} \ (\text{TRN}(n)) \in \text{in} \]

\[ \text{len} \ \text{exgl} = \text{len} \ \text{exl} = \text{len} \ \text{hl} = \text{len} \ \text{lll} + 1 \]

\[ \land \ \text{card} \ \text{xtr}_{GIds}(\text{exgl}) = 2 \ast \text{len} \ \text{exgl} \]

8.4.4.1.5 [e] Toll-gate to Calculator Channels:

We distinguish between entry and exit gates.

Toll road entry and exit gates offers the road pricing calculator a pair: whether it is an entry or an exit gate, and pair of the passing vehicle’s identity and the time-stamped identity of the link associated with the toll-gate to the road pricing calculator via a (gate to calculator) channel.

\[ \text{type} \]

\[ \text{EE} = \text{"entry" | \"exit"} \]

\[ \text{channel} \]

\[ (g_{\text{ch}}[g_{\text{ci}}][g_{\text{gi}} \in \text{gis}]) : \text{EETIVILI} \]
8.4 Domain Requirements

8.4.4.1.6 [f] Road Pricing Calculator Attributes:

396 The road pricing attributes include a programmable traffic map, \( \text{trm:TRM} \), which, for each vehicle inside the toll-road net, records a chronologically ordered list of each vehicle’s timed position, \((\tau),\text{lpos})\), and a static (total) road location function, \( \text{vplf:VPLF} \). The vehicle position location function, \( \text{vplf:VPLF} \), which, given a local position, \( \text{lpos:LPos} \), yields either the simple vehicle position, \( \text{svpos:SVPos} \), designated by the GNSS-provided position, or yields the response that the provided position is off the toll-road net. The \( \text{vplf:VPLF} \) function is constructed, construct_vplf.

397 from awareness, of a geodetic road map, \( \text{GRM} \), of the topology of the extended net, \( \text{n}_\text{c} \times \text{n}_\text{e} \), including the mereology and the geodetic attributes of links and hubs.

398 The geodetic road map maps geodetic locations into hub and link identifiers.

399 A geodetic road map maps geodetic link locations into link identifiers and geodetic hub locations into hub identifiers.

400 We sketch the construction, \( \text{geo:GRM} \), of geodetic road maps.

401 The obtainSVPos function obtains a single vehicle position, \( \text{svpos} \), from a geodetic road map, \( \text{grm:GRM} \), and a local position, \( \text{lpos} \): \( \text{geo:GRM} : \text{N} \rightarrow \text{GRM} \) and \( \text{obtainSVPos: GRM} \rightarrow \text{LPos} \rightarrow \text{SVPos} \) where \( \text{within}(\text{lpos},\text{grm}(\text{li})) \) and \( \text{within}(\text{lpos},\text{grm}(\text{hi})) \) are a predicate which holds if one argument is a point set representation of the geodetic locations of a link or a hub. The design of the \( \text{obtainSVPos} \) represents an interesting challenge.

8.4.4.1.7 [g] “Total” System State:

Global values:

402 There is a given domain, \( \delta: \Delta: \text{C} \); 
403 there is the net, \( \text{n}_\text{c}: \text{N}_\text{c} \), of that domain; 
404 there is toll-road net, \( \text{trn}_\text{c}: \text{TRN}_\text{c} \), of that net; 
405 there is a set, \( \text{egs}_\text{c}: \text{EG}_\text{c} \), of entry gates; 
406 there is a set, \( \text{xgs}_\text{c}: \text{XG}_\text{c} \), of exit gates.

407 there is a set, \( \text{gis}_\text{c}: \text{GI}_\text{c} \), of gate identifiers; 
408 there is a set, \( \text{vs}_\text{c}: \text{V}_\text{c} \), of vehicles; 
409 there is a set, \( \text{vis}_\text{c}: \text{VI}_\text{c} \), of vehicle identifiers; 
410 there is the road-pricing calculator, \( \text{c}_\text{c}: \text{C}_\text{c} \) and 
411 there is its unique identifier, \( \text{cli}_\text{c}: \text{CL} \).

In the following we shall omit the cumbersome \( \_c \) subscripts.
8.4.4.1.8 [h] “Total” System Behaviour:

The signature and definition of the system behaviour is sketched as are the signatures of the vehicle, toll-gate and road pricing calculator. We shall model the behaviour of the road pricing system as follows: we shall not model behaviours nets, hubs and links; thus we shall model only the behaviour of vehicles, veh, the behaviour of toll-gates, gate, and the behaviour of the road-pricing calculator, calc. The behaviours of vehicles and toll-gates are presented here. But the behaviour of the road-pricing calculator is “deferred” till Sect. 8.5.1.3 since it reflects an interface requirements.

412 The road pricing system behaviour, sys, is expressed as:
   a with the parallel, ||, (distributed) composition of the
   behaviours of all vehicles,
   b with the parallel composition of the parallel
   (likewise distributed) composition of the
   behaviours of all entry gates,
   c with the parallel composition of the parallel
   (likewise distributed) composition of the
   behaviours of all exit gates,
   d with the parallel composition of the behaviour
   of the road-pricing calculator,

value
412 sys: Unit → Unit
412 sys() =
412a || (veh, (mero, v(xg)) → VPLF → VPos)
412b || | (gate, (mero, eg(eg(xg))) → VK → EG)
412c || | (calc, (mero, (xg(xg))) → XG)
412d || calc

413 veh: (ci, GI gis) → in attr, TrIGPos[gi] out v, ci | Unit
419 gate: (ci, VI vis) → exit EE
419a in attr, entry, ch (gi, ci), attr, exit, ch (gi, ci)
419b out attr, barrier, ch (gi, ci), exit, ch (gi, ci)
453 calc: (vi, VPos gis) → VPLF → TRM
453a in (v, vi, ci, gis) | Unit
453b in (v, vi, ci, gis) | Unit

We consider “entry” or “exit” to be a static attribute of toll-gates. The behaviour signatures were determined as per
the techniques presented in Chapters 3–6.

Vehicle Behaviour: We refer to the vehicle behaviour, in the domain, described in Sect. 8.2’s The Road Traffic System Behaviour Items 331 and Items 332, Page 206 and, projected, Page 215.

413 Instead of moving around by explicitly expressed internal non-determinism\(^\text{18}\) vehicles move around
by unstated internal non-determinism and instead receive their current position from the global positioning
subsystem.

414 At each moment the vehicle receives its time-stamped global position, (τ, lpos): TiGPos,
415 from which it calculates the local position, lpos: VPos
416 which it then communicates, with its vehicle identification, (vi, (τ, lpos)), to the road pricing subsystem —
417 whereupon it resumes its vehicle behaviour.

value
413 veh: (ci, GI gis) →
413a in attr, TiGPos ch, vi | out v, ci | Unit
413b veh | Unit
414 let (τ, lpos) = attr, TiGPos ch, vi? in
415 let lpos = loc, pos, lpos, ci in
416 v, ci | ! (vi, (τ, lpos))
417 veh | Unit
417a pre vi ∈ vis

The vehicle signature has attr, TiGPos ch, vi model an external vehicle attribute and v, ch, vi, ci the embedded attribute sharing [70, Sect. 4.1.1 and 4.5.2] between vehicles (their position) and the price calculator’s road map. The above behaviour represents an assumption about the behaviour of vehicles. If
we were to design software for the monitoring and control of vehicles then the above vehicle behaviour would have to be refined in order to serve as a proper interface requirements. The refinement would include handling concerns about the drivers’ behaviour when entering, passing and exiting toll-gates, about the proper function of the GNSS equipment, and about the safe communication with the road price calculator. The above concerns would already have been addressed in a model of domain facets such as human behaviour, technology support, proper tele-communications scripts, et cetera. We refer to Chapter 7.

**Gate Behaviour:** The entry and the exit gates have “vehicle enter”, “vehicle exit” and “timed vehicle identification” sensors. The following assumption can now be made: during the time interval between a gate’s “entry” sensor having first sensed a vehicle entering that gate and that gate’s “exit” sensor having last sensed that vehicle leaving that gate that gate’s vehicle time and “identify” sensor registers the time when the vehicle is entering the gate and that vehicle’s unique identification. We sketch the toll-gate behaviour:

418 We parameterise the toll-gate behaviour as either an entry or an exit gate. 419 Toll-gates operate autonomously and cyclically.
420 The attr\_enter\_ch event “triggers” the behaviour specified in formula line Item 421 starting with a “Raise” barrier action.
421 The time-of-passing and the identity of the passing vehicle is sensed by attr\_passing\_ch channel events.
422 Then the road pricing calculator is informed of time-of-passing and of the vehicle identity vi and the link li associated with the gate – and with a “Lower” barrier action.
423 And finally, after that vehicle has left the entry or exit gate the barrier is again “Lower”ed and
424 that toll-gate’s behaviour is resumed.

The gate signature’s attr\_enter\_ch[gi], attr\_passing\_ch[gi], attr\_barrier\_ch[gi] and attr\_leave\_ch[gi] model respective external attributes [70, Sect. 4.1.1 and 4.5.2] (the attr\_barrier\_ch[gi] models reactive (i.e., output) attribute), while g\_c\_ch[gi,ci] models the embedded attribute sharing between gates (their identification of vehicle positions) and the calculator road map. The above behaviour represents an assumption about the behaviour of toll-gates. If we were to design software for the monitoring and control of toll-gates then the above gate behaviour would have to be refined in order to serve as a proper interface requirements. The refinement would include handling concerns about the drivers’ behaviour when entering, passing and exiting toll-gates, about the proper function of the entry, passing and exit sensors, about the proper function of the gate barrier (opening and closing), and about the safe communication with the road price calculator. The above concerns would already have been addressed in a model of domain facets such as human behaviour, technology support, proper tele-communications scripts, et cetera.
We shall define the calculator behaviour in Sect. 8.5.1.3 on Page 233. The reason for this deferral is that it exemplifies interface requirements.

8.4.4.2 Discussion

The requirements assumptions expressed in the specifications of the vehicle and gate behaviours assume that these behave in an orderly fashion. But they seldom do! The \texttt{attr\_TIGPos\_ch} sensor may fail. And so may the \texttt{attr\_enter\_ch}, \texttt{attr\_passing\_ch}, and \texttt{attr\_leave\_ch} sensors and the \texttt{attr\_barrier\_ch} actuator. These attributes represent support technology facets. They can fail. To secure fault tolerance one must prescribe very carefully what counter-measures are to be taken and/or the safety assumptions. We refer to [380, 256, 301]. They cover three alternative approaches to the handling of fault tolerance. Either of the approaches can be made to fit with our approach. First one can pursue our approach to where we stand now. Then we join the approaches of either of [380, 256, 301]. [256] likewise decompose the requirements prescription as is suggested here.

8.4.5 Requirements Fitting

Often a domain being described “fits” onto, is “adjacent” to, “interacts” in some areas with, another domain: transportation with logistics, health-care with insurance, banking with securities trading and/or insurance, and so on. The issue of requirements fitting arises when two or more software development projects are based on what appears to be the same domain. The problem then is to harmonise the two or more software development projects by harmonising, if not too late, their requirements developments.

We thus assume that there are \(n\) domain requirements developments, \(d_1, d_2, \ldots, d_n\), being considered, and that these pertain to the same domain — and can hence be assumed covered by a same domain description.

**Definition: 85 Requirements Fitting:** By requirements fitting we mean a harmonisation of \(n > 1\) domain requirements that have overlapping (shared) not always consistent parts and which results in \(n\) partial domain requirements, \(p_1, p_2, \ldots, p_n\), and \(m\) shared domain requirements, \(s_1, s_2, \ldots, s_m\), that “fit into” two or more of the partial domain requirements. The above definition pertains to the result of “fitting.” The next definition pertains to the act, or process, of “fitting”.

**Definition: 86 Requirements Harmonisation:** By requirements harmonisation we mean a number of alternative and/or co-ordinated prescription actions, one set for each of the domain requirements actions: Projection, Instantiation, Determination and Extension. They are – we assume \(n\) separate software product requirements: Projection: If the \(n\) product requirements do not have the same projections, then identify a common projection which they all share, and refer to it as the common projection. Then develop, for each of the \(n\) product requirements, if required, a specific projection of the common one. Let there be \(m\) such specific projections, \(m \leq n\). Instantiation: First instantiate the common projection, if any instantiation is needed. Then for each of the \(m\) specific projections instantiate these, if required. Determination: Likewise, if required, “perform” “determination” of the possibly instantiated common projection, and, similarly, if required, “perform” “determination” of the up to \(m\) possibly instantiated projections. Extension: Finally “perform extension” likewise: First, if required, of the common projection (etc.), then, if required, on the up \(m\) specific projections (etc.). These harmonization developments may possibly interact and may need to be iterated.

By a partial domain requirement we mean a domain requirement which is short of (that is, is missing) some prescription parts: text and formula. By a shared domain requirement we mean a domain requirements. By requirements fitting \(m\) shared domain requirements texts, \(ssdrs\), into \(n\) partial domain requirements we mean that there is for each partial domain requirements, \(pdr_i\), an identified, non-empty subset of \(ssdrs\) (could be all of \(ssdrs\)), \(ssdr_i\), such that textually conjoining \(ssdr_i\) to \(pdr_i\), i.e., \(ssdrs \uplus pdr_i\) can be claimed to yield the “original” \(d_i\), that is, \(\mathcal{M}(ssdrs \uplus pdr_i) \subseteq \mathcal{M}(d_i)\), where \(\mathcal{M}\) is a suitable meaning function over prescriptions.
8.5 Interface and Derived Requirements

We remind the reader that interface requirements can be expressed only using terms from both the domain and the machine. Users are not part of the machine. No reference can be made to users, such as “the system must be user friendly”, and the like! By interface requirements we [also] mean requirements prescriptions which refines and extends the domain requirements by considering those requirements of the domain requirements whose endurants (parts, materials) and perdurants (actions, events and behaviours) are “shared” between the domain and the machine (being requirements prescribed). The two interface requirements definitions above go hand-in-hand, i.e., complement one-another.

By derived requirements we mean requirements prescriptions which are expressed in terms of the machine concepts and facilities introduced by the emerging requirements.

8.5.1 Interface Requirements

8.5.1.1 Shared Phenomena

By sharing we mean (a) that some or all properties of an endurant is represented both in the domain and “inside” the machine, and that their machine representation must at suitable times reflect their state in the domain; and/or (b) that an action requires a sequence of several “on-line” interactions between the machine (being requirements prescribed) and the domain, usually a person or another machine; and/or (c) that an event arises either in the domain, that is, in the environment of the machine, or in the machine, and need be communicated to the machine, respectively to the environment; and/or (d) that a behaviour is manifested both by actions and events of the domain and by actions and events of the machine. So a systematic reading of the domain requirements shall result in an identification of all shared endurants, parts and materials; and perdurants actions, events and behaviours. Each such shared phenomenon shall then be individually dealt with: endurant sharing shall lead to interface requirements for data initialisation and refreshment as well as for access to endurant attributes; action sharing shall lead to interface requirements for interactive dialogues between the machine and its environment; event sharing shall lead to interface requirements for how such event are communicated between the environment of the machine and the machine; and behaviour sharing shall lead to interface requirements for action and event dialogues between the machine and its environment.

So how do we cope with the statement: “the system must be user friendly”? We refer to Sect. 8.5.3.2 on Page 237 for a discussion of this issue.
8.5.1.1.1 Environment–Machine Interface:

Domain requirements extension. Sect. 8.4.4, usually introduce new endurants into (i.e., ‘extend’ the) domain. Some of these endurants may become elements of the domain requirements. Others are to be projected “away”. Those that are let into the domain requirements either have their endurants represented, somehow, also in the machine, or have (some of) their properties, usually some attributes, accessed by the machine. Similarly for perdurants. Usually the machine representation of shared perdurants access (some of) their properties, usually some attributes. The interface requirements must spell out which domain extensions are shared. Thus domain extensions may necessitate a review of domain projection, instantiations and determination. In general, there may be several of the projection–eliminated parts (etc.) whose dynamic attributes need be accessed in the usual way, i.e., by means of attr\_XYZ\_ch channel communications (where XYZ is a projection–eliminated part attribute).

Example 123 We refer to Fig. 8.2 on Page 223. We do not represent the GNSS system in the machine: only its “effect”: the ability to record global positions by accessing the GNSS attribute (channel):

channel 372 \{attr\_TiGPos\_ch[vⁱ][vⁱ:VI vⁱ ∈ xtr\_Vls(vs)]: TiGPos\}

And we do not really represent the gate nor its sensors and actuator in the machine. But we do give an idealised description of the gate behaviour, see Items 419–424 Instead we represent their dynamic gate attributes:

- (382) the vehicle entry sensors (leftmost ⚡),
- (382) the vehicle identity sensor (center ⚡), and
- (383) the vehicle exit sensors (rightmost ⚡)

by channels — we refer to Example 121 (Sect. 8.5.1.3, Page 223):

channel 382 \{attr\_entry\_ch[gⁱ][gⁱ:GI xtr\_eGls(trn)] “enter”\}
channel 383 \{attr\_exit\_ch[gⁱ][gⁱ:GI xtr\_xGls(trn)] “exit”\}

8.5.1.2 Shared Endurants

Example 124 The main shared endurants are the vehicles, the net (hubs, links, toll-gates) and the price calculator. As domain endurants hubs and links undergo changes, all the time, with respect to the values of several attributes: length, geodetic information, names, wear and tear (where-ever applicable), last/next scheduled maintenance (where-ever applicable), state and state space, and many others. Similarly for vehicles: their position, velocity and acceleration, and many other attributes. We then come up with something like hubs and links are to be represented as tuples of relations; each net will be represented by a hubs relation and a links relation; each hub and each link may or will be represented by several tuples; et cetera. In this database modeling effort it must be secured that “standard” operations on nets, hubs and links can be supported by the chosen relational database system ⚡

8.5.1.2.1 Data Initialisation:

In general, one must prescribe data initialisation, that is provision for an interactive user interface dialogue with a set of proper display screens, one for establishing net, hub or link attributes names and their types,
and, for example, two for the input of hub and link attribute values. Interaction prompts may be prescribed: next input, on-line vetting and display of evolving net, etc. These and many other aspects may therefore need prescriptions.

**Requirements: Interface Requirements, Shared Endurant Initialisation**

**Example 125** The domain is that of the road net, n:N. By ‘shared road net initialisation’ we mean the “ab initio” establishment, “from scratch”, of a data base recording the properties of all links, l:L, and hubs, h:H, their unique identifications, uid\_L(l) and uid\_H(h), their mereologies, mereo\_L(l) and mereo\_H(h), the initial values of all their static and programmable attributes and the access values, that is, channel designations for all other attribute categories.

There are r\_l and r\_h “recorders” recording link, respectively hub properties – with each recorder having a unique identity.

Each recorder is charged with the recording of a set of links or a set of hubs according to some partitioning of all such.

The recorders inform a central data base, net\_db, of their recordings (ri,hol,(uj,mj,attrs\_j)) where

ri is the identity of the recorder,

hol is either a hub or a link literal,

uj = uid\_L(l) or uid\_H(h) for some link or hub,

mj = mereo\_L(l) or mereo\_H(h) for that link or hub and

attrs\_j are attributes for that link or hub — where attributes is a function which “records” all respective static and dynamic attributes (left undefined).

type
425 RI
value
425 rl,rh:NAT axiom rl>0 ∧ rh>0

type
427 M = RI”\times”LNK”\times”HUB | RI”\times”hub”\times”HUB

427 LNK = LI × HI-set × LATTRS
427 HUB = HI × LI-set × HATTRS

value
426 partitioning: L-set→Nat→(L-set)∗
426 | H-set→Nat→(H-set)∗
426 partitioning(s)(r) as sl
426 post: len sl = r ∪ εlems sl = s
426 ∧ ∀ si,sj:(L-set\|H-set) •
426 si≠{}∧sj≠{}∧\{si,sj\}⊆εlems ss⇒si ∩ sj={} 

433 The r\_l + r\_h recorder behaviours interact with the one net\_db behaviour

canal
433 r\_db: RI×(LNK\|HUB)
value
433 link\_rec: RI → L-set → out r\_db Unit
433 hub\_rec: RI → H-set → out r\_db Unit
433 net\_db: Unit → in r\_db Unit

434 The data base behaviour, net\_db, offers to receive messages from the link and hub recorders.
435 The data base behaviour, net\_db, deposits these messages in respective variables.
436 Initially there is a net, n:N.
from which is observed its links and hubs.
These sets are partitioned into \( r_l \), respectively \( r_h \) length lists of non-empty links and hubs.
The ab-initio data initialisation behaviour, \texttt{ab\_initio\_data}, is then the parallel composition of link recorder, hub recorder and data base behaviours with link and hub recorder being allotted appropriate link, respectively hub sets.
We construct, for technical reasons, as the reader will soon see, disjoint lists of link, respectively hub recorder identities.

\[
\text{value}
\]
\texttt{net\_db:}
\texttt{variable}
\texttt{lnk\_db: (RI} \times \texttt{L}\texttt{-set)}
\texttt{hub\_db: (RI} \times \texttt{H}\texttt{-set)}
\texttt{value}
\texttt{n:N}
\texttt{ls:L\text{-set} = \texttt{obs}\_Ls(\texttt{obs}\_Ls(n))}
\texttt{hs:H\text{-set} = \texttt{obs}\_Hs(\texttt{obs}\_Hs(n))}
\texttt{ls!:\texttt{L}\text{-set}^* = \texttt{partitioning}(\texttt{ls})(\texttt{rl})}
\texttt{lh!:\texttt{H}\text{-set}^* = \texttt{partitioning}(\texttt{hs})(\texttt{rh})}
\texttt{rill:RI} \times \texttt{axiom len} \texttt{rill} = \texttt{rl} = \texttt{card elems rill}
\texttt{rihl:RI} \times \texttt{axiom len} \texttt{rihl} = \texttt{rh} = \texttt{card elems rihl}
\texttt{ab\_initio\_data: Unit} \rightarrow \texttt{Unit}
\texttt{ab\_initio\_data()} \equiv
\texttt{∥\{lnk\_rec(\texttt{rill}[i])(\texttt{ls}[i]):\texttt{Nat} \leq i \leq \texttt{rl}∥}
\texttt{∥\{hub\_rec(\texttt{rihl}[i])(\texttt{lh}[i]):\texttt{Nat} \leq i \leq \texttt{rh}∥}
\texttt{∥\texttt{net\_db}()}$

The link and the hub recorders are near-identical behaviours.
They both revolve around an imperatively stated \texttt{for all} \ldots \texttt{do} \ldots \texttt{end}. The selected link (or hub) is inspected and the “data” for the data base is prepared from
the unique identifier,
the mereology, and
the attributes.
These “data” are sent, as a message, prefixed the senders identity, to the data base behaviour.
We presently leave the \ldots unexplained.

\[
\text{value}
\]
\texttt{link\_rec: RI} \rightarrow \texttt{L\text{-set} \rightarrow Unit}
\texttt{link\_rec(ri,ls) \equiv}
\texttt{for} \ \forall \ l:L\text{-l} \in \texttt{ls do uid}\_L(l)}
\texttt{let lnk = (uid}\_L(l), \texttt{mereo}\_L(l), \texttt{attributes(l)} in}
\texttt{rdb ! (ri,\texttt{"link"},lnk);}
\texttt{... end}
\texttt{end}

\[
\text{hub\_rec: RI} \times \texttt{H\text{-set} \rightarrow Unit}
\texttt{hub\_rec(ri,hs) \equiv}
\texttt{for} \ \forall \ h:H\text{-h} \in \texttt{hs do uid}\_H(h)}
\texttt{let hub = (uid}\_H(h), \texttt{mereo}\_H(h), \texttt{attributes(h)} in
8.5 Interface and Derived Requirements

446 \texttt{rdb ! (ri,”hub”,hub);} \\
447 \texttt{... end} \\
448 \texttt{end}

448 The \texttt{net_db} data base behaviour revolves around a seemingly “never-ending” cyclic process. 
449 Each cycle “starts” with acceptance of some, 
450 either link or hub data. 
451 If link data then it is deposited in the link data base, 
452 if hub data then it is deposited in the hub data base.

\begin{verbatim}
value
448 \texttt{net_db()} ≡ \\
449 \texttt{let (ri,hol,data) = r_db ? in} \\
450 \texttt{case hol of} \\
451 \texttt{“link” → ... ; lnk_db := lnk_db ∪ (ri,data),} \\
452 \texttt{“hub” → ... ; hub_db := hub_db ∪ (ri,data)} \\
450 \texttt{end end ;} \\
448 \texttt{... ;} \\
448 \texttt{net_db()} \\
\end{verbatim}

The above model is an idealisation. It assumes that the link and hub data represent a well-formed net. 
Included in this well-formedness are the following issues: (a) that all link or hub identifiers are commu-
nicated exactly once, (b) that all mereologies refer to defined parts, and (c) that all attribute values lie 
within an appropriate value range. If we were to cope with possible recording errors then we could, for 
example, extend the model as follows: (i) when a link or a hub recorder has completed its recording then it 
increments an initially zero counter (say at formula Item 447); (ii) before the net data base recycles it tests 
whether all recording sessions has ended and then proceeds to check the data base for well-formedness 
issues (a–b–c) (say at formula Item 448')

The above example illustrates the ‘interface’ phenomenon: In the formulas, for example, we show both 
manifest domain entities, viz., \texttt{n, l, h} etc., and abstract (required) software objects, viz., \texttt{(ui, me, attrs)}.

8.5.1.2.2 Data Refreshment:

One must also prescribe data refreshment: an interactive user interface dialogue with a set of proper dis-
play screens one for selecting the updating of net, of hub or of link attribute names and their types and, 
for example, two for the respective update of hub and link attribute values. Interaction-prompts may be 
prescribed: next update, on-line vetting and display of revised net, etc. These and many other aspects may 
therefore need prescriptions.

8.5.1.3 Shared Perdurants

We can expect that for every part in the domain that is shared with the machine and for which there is 
a corresponding behaviour of the domain there might be a corresponding process of the machine. If a 
projected, instantiated, ‘determinated’ and possibly extended domain part is dynamic, then it is definitely 
a candidate for being shared and having an associated machine process.

We now illustrate the concept of shared perdurants via the domain requirements extension example of 
Sect. 8.4.4, i.e. Example 121 Pages 222–227.

Requirements: Interface Requirements, Shared Behaviours

Example 126 Road Pricing Calculator Behaviour:

453 The road-pricing calculator alternates between offering to accept communication from 
454 either any vehicle
or any toll-gate.

\[
\text{calc: ci:CI} \times (\text{vis:VI-set} \times \text{gis:GI-set}) \rightarrow \text{RLF} \rightarrow \text{TRM} \rightarrow
\]

\[
\text{in } \{\text{v_c_ch}[\text{ci,vi}] | \text{vi:VI} \in \text{vis}\},
\]

\[
\{g_c_ch[\text{ci,gi}] | \text{gi:GI} \in \text{gis}\} \quad \text{Unit}
\]

\[
\text{react_to_vehicles}(\text{ci,vis,gis})(\text{rlf})(\text{trm})
\]

\[
\text{react_to_gates}(\text{ci,vis,gis})(\text{rlf})(\text{trm})
\]

\[
\text{pre } \text{ci} = \text{ci}_E \land \text{vis} = \text{vis}_E \land \text{gis} = \text{gis}_E.
\]

The calculator signature’s \(v_c\text{ch}[\text{ci,vi}]\) and \(g_c\text{ch}[\text{ci,gi}]\) model the embedded attribute sharing between vehicles (their position), respectively gates (their vehicle identification) and the calculator road map [70, Sect. 4.1.1 and 4.5.2].

If the communication is from a vehicle inside the toll-road net then its toll-road net position, \(v_p\), is found from the road location function, \(\text{rlf}\), and the calculator resumes its work with the traffic map, \(\text{trm}\), suitably updated, otherwise the calculator resumes its work with no changes.

\[
\text{react_to_vehicles}(\text{ci,vis,gis},v_p)(\text{rlf})(\text{trm})
\]

\[
\text{let } (\text{vi,}(\tau,\text{pos})) = \{\text{v_c_ch}[\text{ci,vi}] | \text{vi:VI} \in \text{vis}\} \quad \text{in}
\]

\[
\text{if } \text{vi} \in \text{domain } \text{trm}
\]

\[
\text{then let } v_p = \text{vplf}(\text{pos}) \quad \text{in}
\]

\[
\text{calc}(\text{ci,vis,gis},v_p)(\text{rlf})(\text{trm}) \left[ \text{vi} \mapsto \langle (\tau, \text{trm}(\text{vi})\rangle \right]
\]

\[
\text{else } \text{calc}(\text{ci,vis,gis},v_p)(\text{trm}) \quad \text{end}
\]

If the communication is from a gate, then that gate is either an entry gate or an exit gate; if it is an entry gate then the calculator resumes its work with the vehicle (that passed the entry gate) now recorded, afresh, in the traffic map, \(\text{trm}\). Else it is an exit gate and the calculator concludes that the vehicle has ended its to-be-paid-for journey inside the toll-road net, and hence to be billed; then the calculator resumes its work with the vehicle now removed from the traffic map, \(\text{trm}\).

\[
\text{react_to_gates}(\text{ci,vis,gis},v_p)(\text{trm})
\]

\[
\text{let } (\text{ee,}(\tau,(\text{vi,li})) = \{\text{g_c_ch}[\text{ci,gi}] | \text{gi:GI} \in \text{gis}\} \quad \text{in}
\]

\[
\text{case } \text{ee} \text{ of }
\]

\[
"\text{Enter}" \rightarrow
\]

\[
\text{calc}(\text{ci,vis,gis},v_p)(\text{trm}) \left[ \text{in} \mapsto \langle (\tau, \text{SonL}(\text{li}))\rangle \right],
\]

\[
"\text{Exit}" \rightarrow
\]

\[
\text{billing}(\text{vi},\text{trm}(\text{vi})\langle (\tau, \text{SonL}(\text{li}))\rangle);
\]

\[
\text{calc}(\text{ci,vis,gis},v_p)(\text{trm}) \left[ \text{vi} \mapsto \langle (\tau, \text{trm}(\text{vi}))\rangle \right]
\]

The above behaviour is the one for which we are to design software.
8.5 Interface and Derived Requirements

8.5.2 Derived Requirements

Definition: Derived Perdurant: By a derived perdurant we shall understand a perdurant which is not shared with the domain, but which focus on exploiting facilities of the software or hardware of the machine.

“Exploiting facilities of the software”, to us, means that requirements, imply the presence, in the machine, of concepts (i.e., hardware and/or software), and that it is these concepts that the derived requirements “rely” on. We illustrate all three forms of perdurant extensions: derived actions, derived events and derived behaviours.

8.5.2.1 Derived Actions

Definition: Derived Action: By a derived action we shall understand (a) a conceptual action (b) that calculates a usually non-Boolean valued property from, and possibly changes to (c) a machine behaviour state (d) as instigated by some actor.

Requirements: Domain Requirements, Derived Action – Tracing Vehicles

Example 127 The example is based on the Road Pricing Calculator Behaviour of Example 126 on Page 233. The “external” actor, i.e., a user of the Road Pricing Calculator system wishes to trace specific vehicles “cruising” the toll-road. That user (a Road Pricing Calculator staff), issues a command to the Road Pricing Calculator system, with the identity of a vehicle not already being traced. As a result the Road Pricing Calculator system augments a possibly void trace of the timed toll-road positions of vehicles.

We augment the definition of the calculator definition Items 453–466, Pages 233–234.

467 Traces are modeled by a pair of dynamic attributes:
   a as a programmable attribute, \( tra:TRA \), of the set of identifiers of vehicles being traced, and
   b as a reactive attribute, \( vdu:VDU \), that maps vehicle identifiers into time-stamped sequences of simple vehicle positions, i.e., as a subset of the \( trm:TRM \) programmable attribute.

468 The actor-to-calculator begin or end trace command, \( cmd:Cmd \), is modeled as an autonomous dynamic attribute of the calculator.

469 The calculator signature is furthermore augmented with the three attributes mentioned above.

470 The occurrence and handling of an actor trace command is modeled as a non-deterministic external choice and a react_to_trace_cmd behaviour.

471 The reactive attribute value \( \text{attr}_vdu,ch ? \) is that subset of the traffic map \( trm \) which records just the time-stamped sequences of simple vehicle positions being traced \( trm \).

467a \( TRA = VI-set \)

467b \( VDU = TRM \)

468 \( Cmd = BTr | ETr \)

468 \( BTr :: VI \)

468 \( ETr :: VI \)

469 \( \text{calc}: ci:CI \times \{ \text{vis}:VI-set \times \text{gis}:GI-set \} \rightarrow \text{RLF} \rightarrow \text{TRM} \rightarrow \text{TRA} \)

454,455 \( \text{in} \{ v_{ci,vi} : ci,vi | vi:VI \text{ vi} \in \text{vis} \}, \)

454,455 \( \{ g_{ci,gi} : ci,gi | gi:GI \text{ gi} \in \text{gis} \}, \)

470,471 \( \text{attr}_cmd,ch,\text{attr}_vdu,ch \text{ Unit} \)

453 \( \text{calc}(ci,(\text{vis},gis))(\text{RLF})(\text{TRM})(\text{TRA}) \equiv \)
REQUIREMENTS

react_to_vehicles(ci,(vis,gis))(r)(trm)(tra)

react_to_gates(ci,(vis,gis))(r)(trm)(tra)

react_to_trace_cmd(ci,(vis,gis))(r)(trm)(tra)

pre ci = ciE ∧ vis = visE ∧ gis = gisE

axiom □ attr_vdu_ch[ci]? = trm|tra

The 470,471 attr_cmd_ch,attr_vdu_ch of the calculator signature models the calculator’s external command and visual display unit attributes.

The react_to_trace_cmd alternative behaviour is either a ”Begin” or an ”End” request which identifies the affected vehicle.

If it is a ”Begin” request and the identified vehicle is already being traced then we do not prescribe what to do ! Else we resume the calculator behaviour, now recording that vehicle as being traced.

If it is an ”End” request and the identified vehicle is already being traced then we do not prescribe what to do ! Else we resume the calculator behaviour, now recording that vehicle as no longer being traced.

react_to_trace_cmd(ci,(vis,gis))(vplf)(trm)(tra) ≡

case attr_cmd_ch[ci]? of

mkBTr(vi) → if vi ∈ tra then chaos

else calc(ci,(vis,gis))(vplf)(trm)(tra ∪ {vi}) end

mkETr(vi) → if vi /nelement tra then chaos

else calc(ci,(vis,gis))(vplf)(trm)(tra\ {vi}) end

end

The above behaviour, Items 453–478, is the one for which we are to design software ■

Example 127 exemplifies an action requirement as per definition 88: (a) the action is conceptual, it has no physical counterpart in the domain; (b) it calculates (471) a visual display (vdu); (c) the vdu value is based on a conceptual notion of traffic road maps (trm), an element of the calculator state; (d) the calculation is triggered by an actor (attr_cmd_ch).

8.5.2.2 Derived Events

Definition: 89 Derived Event: By a derived event we shall understand (a) a conceptual event, (b) that calculates a property or some non-Boolean value (c) from a machine behaviour state change ■

Requirements: Domain Requirements, Derived Event, Current Maximum Flow

Example 128 The example is based on the Road Pricing Calculator Behaviour of Examples 127 and 126 on Page 233. By “the current maximum flow” we understand a time-stamped natural number, the number representing the highest number of vehicles which at the time-stamped moment cruised or now cruises around the toll-road net. We augment the definition of the calculator definition Items 453–478, Pages 233–236.

479 We augment the calculator signature with

480 a time-stamped natural number valued dynamic programmable attribute, (t:T,max:Max).

481 Whenever a vehicle enters the toll-road net, through one of its [entry] gates,

a it is checked whether the resulting number of vehicles recorded in the road traffic map is higher than the hitherto maximum recorded number.
b If so, that programmable attribute has its number element “upped” by one.
c Otherwise not.

482 No changes are to be made to the react\textsubscript{to}gates behaviour (Items 455–466 Page 234) when a vehicle exits the toll-road net.

The above behaviour, Items 453 on Page 233 through 481c, is the one for which we are to design software.

Example 128 exemplifies a derived event requirement as per Definition 89: (a) the event is conceptual, it has no physical counterpart in the domain; (b) it calculates (481b) the max value based on a conceptual notion of traffic road maps (trm), (c) which is an element of the calculator state.

8.5.2.3 No Derived Behaviours

There are no derived behaviours. The reason is as follows. Behaviours are associated with parts. A possibly ‘derived behaviour’ would entail the introduction of an ‘associated’ part. And if such a part made sense it should – in all likelihood – already have been either a proper domain part or become a domain extension. If the domain–to-requirements engineer insist on modeling some interface requirements as a process then we consider that a technical matter, a choice of abstraction.

8.5.3 Discussion

8.5.3.1 Derived Requirements

Formulation of derived actions or derived events usually involves technical terms not only from the domain but typically from such conceptual ‘domains’ as mathematics, economics, engineering or their visualisation. Derived requirements may, for some requirements developments, constitute “sizable” requirements compared to “all the other” requirements. For their analysis and prescription it makes good sense to first having developed “the other” requirements: domain, interface and machine requirements. The treatment of the present chapter does not offer special techniques and tools for the conception, &c., of derived requirements. Instead we refer to the seminal works of [134, 264, 355].

8.5.3.2 Introspective Requirements

Humans, including human users are, in this chapter, considered to never be part of the domain for which a requirements prescription is being developed. If it is necessary to involve humans in the domain description or the requirements prescription then their prescription is to reflect assumptions upon whose behaviour the machine rely. It is therefore that we, above, have stated, in passing, that we cannot accept requirements of the kind: “the machine must be user friendly”; because, in reality, it means “the user must rely upon the machine being ‘friendly’” whatever that may mean. We are not requirements prescribing humans, nor their sentiments!
8.6 Machine Requirements

Other than listing a sizable number of machine requirement facets we shall not cover machine requirements in this chapter. The reason for this is as follows. We find, cf. [41, Sect. 19.6], that when the individual machine requirements are expressed then references to domain phenomena are, in fact, abstract references, that is, they do not refer to the semantics of what they name. Hence machine requirements “fall” outside the scope of this chapter with that scope being “derivation” of requirements from domain specifications with emphasis on derivation techniques that relate to various aspects of the domain.

(A) There are the technology requirements of (1) performance and (2) dependability. Within dependability requirements there are (a) accessibility, (b) availability, (c) integrity, (d) reliability, (e) safety, (f) security and (g) robustness requirements. A proper treatment of dependability requirements need a careful definition of such terms as failure, error, fault, and, from these dependability. (B) And there are the development requirements of (i) process, (ii) maintenance, (iii) platform, (iv) management and (v) documentation requirements. Within maintenance requirements there are (ii.1) adaptive, (ii.2) corrective, (ii.3) perfective, (ii.4) preventive, and (ii.5) extensional requirements. Within platform requirements there are (iii.1) development, (iii.2) execution, (iii.3) maintenance, and (iii.4) demonstration platform requirements. We refer to [41, Sect. 19.6] for an early treatment of machine requirements.

8.7 Summary

8.7.1 Method Principles, Techniques and Tools

Recall that by a method we shall understand a set of principles for selecting and applying a set of techniques using a set of tools in order to construct an artefact.

8.7.1.1 Principles of Requirements

Some of the principles applied in “deriving” requirements prescriptions from domain descriptions are:

**Divide & Conquer**: The separation into

- domain,  
- interface and  
- machine

requirements is an example of ‘divide & conquer’, as is their treatment in the order listed.

**Refinement**: “By and large” we see the ‘transformation’ of domain descriptions into requirements prescriptions as a refinement though with some exceptional ‘deviations’. Instantiation is not a refinement. Determination is. When we say ‘by and large’ we mean “when everything about a situation is considered together”. That is, not all the transformations of this chapter are refinements.

**Conservative Extension**: The extension(s), in our examples, in this chapter is/are an example of conservative extension(s), But there could be other forms of domain extensions which would not be conservative.

8.7.1.2 Techniques of Requirements

The basic technique, in all steps of domain and interface requirements, involve reconsidering the domain sorts and types, then their well-formedness, then their merologies, et cetera. Further techniques, i.e., sub-techniques derive from that.

8.7.1.3 Tools of Requirements

The tools are the usual ones: informal, but disciplined narratives that are ‘fitted’ closely to the formalisations.
8.7.2 Concluding Review

We conclude by briefly reviewing what has been achieved, present shortcomings & possible research challenges, and a few words on relations to “classical requirements engineering”.

8.7.2.1 What has been Achieved?

We have shown how to systematically “derive” initial aspects of requirements prescriptions from domain descriptions. The stages\(^{21}\) and steps\(^{22}\) of this “derivation” are new. We claim that current requirements engineering approaches, although they may refer to a or the ‘domain’, are not really ‘serious’ about this: they do not describe the domain, and they do not base their techniques and tools on a reasoned understanding of the domain. In contrast we have identified, we claim, a logically motivated decomposition of requirements into three phases, cf. Footnote 21., of domain requirements into five steps, cf. Footnote 22 (Page 239), and of interface requirements, based on a concept of shared entities, tentatively into (\(\alpha\)) shared endurants, (\(\beta\)) shared actions, (\(\gamma\)) shared events, and (\(\delta\)) shared behaviours (with more research into the (\(\alpha-\delta\)) techniques needed).

8.7.2.2 Present Shortcomings and Research Challenges

We see three shortcomings: (1) The “derivation” techniques have yet to consider “extracting” requirements from domain facet descriptions. Only by including domain facet descriptions can we, in “deriving” requirements prescriptions, include failures of, for example, support technologies and humans, in the design of fault-tolerant software. (2) The “derivation” principles, techniques and tools should be given a formal treatment. (3) There is a serious need for relating the approach of the present chapter to that of the seminal text book of [355, Axel van Lamsweerde]. [355] is not being “replaced” by the present work. It tackles a different set of problems. We refer to the penultimate paragraph before the Acknowledgment closing.

8.7.2.3 Comparison to “Classical” Requirements Engineering:

Except for a few, represented by two, we are not going to compare the contributions of the present chapter with published journal or conference papers on the subject of requirements engineering. The reason for this is the following. The present chapter, rather completely, we claim, reformulates requirements engineering, giving it a ‘foundation’, in domain engineering, and then developing requirements engineering from there, viewing requirements prescriptions as “derived” from domain descriptions. We do not see any of the papers, except those reviewed below [256] and [134], referring in any technical sense to ‘domains’ such as we understand them.

8.7.2.3.1 [256, Deriving Specifications for Systems That Are Connected to the Physical World]

The paper that comes closest to the present chapter in its serious treatment of the [problem] domain as a Precursor for Requirements development is that of [256, Jones, Hayes & Jackson]. A purpose of [256] (Sect. 1.1, Page 367, last §) is to see “how little can one say” (about the problem domain) when expressing assumptions about requirements. This is seen by [256] (earlier in the same paragraph) as in contrast to our form of domain modeling. [256] reveals assumptions about the domain when expressing rely guarantees in tight conjunction with expressing the guarantee (requirements). That is, analysing and expressing requirements, in [256], goes hand-in-hand with analysing and expressing fragments of the domain. The current chapter takes the view that since, as demonstrated in [70], it is possible to model sizable aspects of domains, then it would be interesting to study how one might “derive” — and which — requirements prescriptions from domain descriptions; and having demonstrated that (i.e., the “how much can be derived”) it seems of

\(^{21}\) (a) domain, (b) interface and (c) machine requirements

\(^{22}\) For domain requirements: (i) projection, (ii) instantiation, (iii) determination, (iv) extension and (v) fitting; etc.

\(^{23}\) We use double quotation marks: “…” to indicate that the derivation is not automatable.
scientific interest to see how that new start (i.e., starting with a priori given domain descriptions or starting with first developing domain descriptions) can be combined with existing approaches, such as [256]. We do appreciate the “tight coupling” of rely–guarantees of [256]. But perhaps one looses understanding the domain due to its fragmented presentation. If the ‘relies’ are not outright, i.e., textually directly expressed in our domain descriptions, then they obviously must be provable properties of what our domain descriptions express. Our, i.e., the present, chapter — with its background in Chapters 3–6 and [70, Sect. 4.7] — develops — with a background in [252, M.A. Jackson] — a set of principles and techniques for the access of attributes. The “discovery” of the CM and SG channels of [256] and of the type of their messages, seems, compared to our approach, less systematic. Also, it is not clear how the [256] case study “scales” up to a larger domain. The sluice gate of [256] is but part of a large (“irrigation”) system of reservoirs (water sources), canals, sluice gates and the fields (water sinks) to be irrigated. We obviously would delineate such a larger system and research & develop an appropriate, both informal, a narrative, and formal domain description for such a class of irrigation systems based on assumptions of precipitation and evaporation. Then the users’ requirements, in [256], that the sluice gate, over suitable time intervals, is open 20% of the time and otherwise closed, could now be expressed more pertinent ly, in terms of the fields being appropriately irrigated.

8.7.2.3.2 [134, Goal-directed Requirements Acquisition]

outlines an approach to requirements acquisition that starts with fragments of domain description. The domain description is captured in terms of predicates over actors, actions, events, entities and (their) relations. Our approach to domain modeling differs from that of [134] as follows: Agents, actions, entities and relations are, in [134], seen as specialisations of a concept of objects. The nearest analogy to relations, in [70], as well as in this chapter, is the signatures of perdurants. Our ‘agents’ relate to discrete endurants, i.e., parts, and are the behaviours that evolve around these parts: one agent per part! [134] otherwise include describing parts, relations between parts, actions and events much like [70] and this chapter does. [134] then introduces a notion of goal. A goal, in [134], is defined as “a nonoperational objective to be achieved by the desired system. Nonoperational means that the objective is not formulated in terms of objects and actions “available” to some agent of the system” [24]

[134] then goes on to exemplify goals. In this, the current chapter, we are not considering goals, also a major theme of [355].

Constraints are operational objectives to be achieved by the desired system. . . . Constraints are operational objectives to be achieved by the desired (i.e., required) system. . . . A constraint operationalising a goal amounts to some abstract “implementation” of this goal” [134]. [134] then goes on to express goals and constraints operationalising these. [134] is a fascinating paper as it shows how to build goals and constraints on domain description fragments.

These papers, [256] and [134], as well as the current chapter, together with such seminal monographs as [380, 301, 355], clearly shows that there are many diverse ways in which to achieve precise requirements

---

24 We have reservations about this definition: Firstly, it is expressed in terms of some of the “things” it is not! (To us, not a very useful approach.) Secondly, we can imagine goals that are indeed formulated in terms of objects and actions ‘available’ to some agent of the system. For example, wrt. the ongoing library examples of [134], the system shall automate the borrowing of books, et cetera. Thirdly, we assume that by “‘available’ to some agent of the system” is meant that these agents, actions, entities, etc., are also required.

25 An example of a goal — for the road pricing system — could be that of shortening travel times of motorists, reducing gasoline consumption and air pollution, while recouping investments on toll-road construction. We consider techniques for ensuring the above kind of goals “outside” the realm of computer & computing science but “inside” the realm of operations research (OR) — while securing that the OR models are commensurate with our domain models.

26 In this chapter we do not exemplify goals, let alone the use of temporal logic. We cannot exemplify all aspects of domain description and requirements prescription, but, if we were, would then use the temporal logic of [380, The Duration Calculus].

27 — that might, however, warrant a complete rewrite.
prescriptions. The [380, 301] monographs primarily study the $\mathcal{D}, \mathcal{S} \models \mathcal{R}$ specification and proof techniques from the point of view of the specific tools of their specification languages\[28\]. Physics, as a natural science, and its many engineering 'renditions', are manifested in many separate sub-fields: Electricity, mechanics, statics, fluid dynamics — each with further sub-fields. It seems, to this author, that there is a need to study the [380, 301, 355] approaches and the approach taken in this chapter in the light of identifying sub-fields of requirements engineering. The title of the present chapter suggests one such sub-field.

8.8 Bibliographical Notes

I have thought about domain engineering for more than 20 years. But serious, focused writing only started to appear since [41, Part IV] — with [37, 34] being exceptions: [43] suggests a number of domain science and engineering research topics; [52] covers the concept of domain facets; [82] explores compositionality and Galois connections. [44, 81] show how to systematically, but, of course, not automatically, “derive” requirements prescriptions from domain descriptions; [56] takes the triptych software development as a basis for outlining principles for believable software management; [48, 61] presents a model for Stanisław Leśniewski’s [115] concept of mereology; [53, 57] present an extensive example and is otherwise a precursor for the present chapter; [58] presents, based on the TripTych view of software development as ideally proceeding from domain description via requirements prescription to software design, concepts such as software demos and simulators; [59] analyses the TripTych, especially its domain engineering approach, with respect to [276, 277, Maslow]’s and [313, Peterson’s and Seligman’s]’s notions of humanity: how can computing relate to notions of humanity; the first part of [62] is a precursor for [70] with the second part of [62] presenting a first formal model of the elicitation process of analysis and description based on the prompts more definitively presented in the current chapter; and with [63] focus on domain safety criticality.

8.9 Exercise Problems

8.9.1 Research Problems

**Exercise 1. A Research Challenge. Bridge to The Hayes-Jackson-Jones Approach:** We refer to Pages 239–240. Study [256, Hayes, Jackson and Jones] and related papers. Then suggest ways and means to incorporate their, the HJJ approach, with that of ours. Let either the HJJ be determining the approach sequence or that of ours.

**Exercise 2. A Research Challenge. Bridge to The Lamsweerde Approach:** We refer to Pages 240–240. Study [134, van Lamsweerde et al.] and related papers. Then suggest ways and means to incorporate their, the KAOS approach, with that of ours. Let either the KAOS be determining the approach sequence or that of ours.

**Exercise 3. A Research Challenge. Bridge to Cyber-Physical Computing Systems:** Study [301, Olderog and Dierks] and related papers. Then suggest ways and means to incorporate their, the Olderog et al. approach, with that of ours. Let either the Olderog et al. be determining the approach sequence or that of ours.

8.9.2 Term Projects

We continue the term projects of Sects. 3.24.3 on Page 81, 4.12.3 on Page 121, 6.14.3 on Page 161, and 7.11.2 on Page 193.

The students are to identify and analyse & describe at least three distinct requirements aspects of their chosen domain:

\[28\] The Duration Calculus [DC], respectively DC, Timed Automata and Z
• domain requirements: projections, instantiations, determinations and extension; and
• interface requirements.

**Exercise 47** An MSc Student Exercise. **The Consumer Market, Requirements:** We refer to Exercise 4 on Page 82, 20 on Page 121, 31 on Page 161, and 40 on Page 193.

**Exercise 48** An MSc Student Exercise. **Financial Service Industry, Requirements:** We refer to Exercise 5 on Page 82, 21 on Page 121, 32 on Page 161, and 41 on Page 193.

**Exercise 49** An MSc Student Exercise. **Container Line Industry, Requirements:** We refer to Exercise 6 on Page 82, 22 on Page 121, 33 on Page 161, and 42 on Page 193.

**Exercise 50** An MSc Student Exercise. **Railway Systems, Requirements:** We refer to Exercise 7 on Page 82, 23 on Page 121, 34 on Page 161, and 43 on Page 193.

**Exercise 51** An MSc Student Exercise. **Part-Material Conjoins: Pipelines, Requirements:** We refer to Appendix Chapter A.

**Exercise 52** A PhD Student Problem. **Part-Material Conjoins: Canals, Requirements:** We refer to Exercise 8 on Page 82, 24 on Page 121, 35 on Page 161, and 44 on Page 193.

**Exercise 53** A PhD Student Problem. **Part-Material Conjoins: Rum Production, Requirements:** We refer to Exercises 9 on Page 82, 25 on Page 121, 36 on Page 161 and 45 on Page 193.

**Exercise 54** A PhD Student Problem. **Part-Materials Conjoins: Waste Management, Requirements:** We refer to Exercise 10 on Page 82, 26 on Page 122, 37 on Page 161, and 46 on Page 193.
Part IV

CLOSING
DEMOS, SIMULATORS, MONITORS AND CONTROLLERS

In this chapter\(^1\) we muse over concepts of demos, simulators, monitors and controllers.

9.1 Introduction

We sketch some observations of the concepts of domain, requirements and modeling – where abstract interpretations of these models cover both a priori, a posteriori and real-time aspects of the domain as well as 1–1 (i.e., real-time), microscopic and macroscopic simulations, real-time monitoring and real-time monitoring & control of that domain. The reference frame for these concepts are domain models: carefully narrated and formally described domains. On the basis of a familiarising example\(^2\) of a domain description, we survey more-or-less standard ideas of verifiable software developments and conjecture software product families of demos, simulators, monitors and monitors & controllers – but now these “standard ideas” are recast in the context of core requirements prescriptions being “derived” from domain descriptions. A background setting for this chapter is the concern for (α) professionally developing the right software, i.e., software which satisfies users expectations, and (ω) software that is right: i.e., software which is correct with respect to user requirements and thus has no “bugs”, no “blue screens”. The present chapter must be seen on the background of a main line of experimental research around the topics of domain science & engineering and requirements engineering and their relation. We refer to earlier chapters of this monograph.

9.1.0.0.1 “Confusing Demos”:

This author has had the doubtful honour, on his many visits to computer science and software engineering laboratories around the world, to be presented, by his colleagues’ aspiring PhD students, so-called demos of “systems” that they were investigating. There always was a tacit assumption, namely that the audience, i.e., me, knew, a priori, what the domain “behind” the “system” being “demo’ed” was. Certainly, if there was such an understanding, it was brutally demolished by the “demo” presentation. My questions, such as “what are you demo’ing” (et cetera) went unanswered. Instead, while we were waiting to see “something interesting” to be displayed on the computer screen we were witnessing frantic, sometimes failed, input of commands and data, “nervous” attempts with “mouse” clickings, etc. – before something intended was displayed. After a, usually 15 minute, grace period, it was time, luckily, to proceed to the next “demo”.

9.1.0.0.2 Aims & Objectives:

The aims of this chapter is to present (a) some ideas about software that either “demo”, simulate, monitor or monitor & control domains; (b) some ideas about “time scaling”: demo and simulation time versus domain time; and (c) how these kinds of software relate. The (undoubtedly very naïve) objectives of the chapter is also to improve the kind of demo-presentations, alluded to above, so as to ensure that the basis for such demos is crystal clear from the very outset of research & development, i.e., that domains be well-described. The chapter, we think, tackles the issue of so-called “model-oriented (or model-based) software development” from altogether different angles than usually promoted.

---

1. This chapter is an edited rendition of [58]
2. – take that of Chapter 3
9.1.0.0.3 **An Exploratory Chapter:**

The chapter is exploratory. There will be no theorems and therefore there will be no proofs. We are presenting what might eventually emerge into $\alpha$ a theory of domains, i.e., a domain science and $\beta$ a software development theory of domain engineering versus requirements engineering.

The chapter is not a “standard” research chapter: it does not compare its claimed achievements with corresponding or related achievements of other researchers – simply because we do not claim “achievements” which have been reasonably well formalised. But we would suggest that you might find some of the ideas of the chapter (in Sect. 9.2) worthwhile. Hence the “divertimento” suffix to the chapter title.

9.1.0.0.4 **Structure of Chapter:**

The structure of the chapter is as follows. In Sect. 9.2 we then outline a series of interpretations of domain descriptions. These arise, when developed in an orderly, professional manner, from requirements prescriptions which are themselves orderly developed from the domain description. The essence of Sect. 9.2 is (i) the (albeit informal) presentation of such tightly related notions as demos (Sect. 9.2.1), simulators (Sect. 9.2.2 on the next page), monitors (Sect. 9.2.3.1 on Page 250) and monitors & controllers (Sect. 9.2.3.2 on Page 250) (these notions can be formalised), and (ii) the conjectures on a product family of domain-based software developments (Sect. 9.2.5 on Page 251). A notion of script-based simulation extends demos and is the basis for monitor and controller developments and uses. The scripts used in our examples are related to time, but one can define non-temporal scripts – so the “carrying idea” of Sect. 9.2 extends to a widest variety of software. We claim that Sect. 9.2 thus brings these new ideas: a tightly related software engineering concept of demo-simulator-monitor-controller machines, and an extended notion of reference models for requirements and specifications [185].

9.2 **Interpretations**

In this main section of the chapter we present a number of interpretations of rôles of domain descriptions.

9.2.1 **What Is a Domain-based Demo?**

A domain-based demo is a software system which “present” endurants and perdurants: actions, events and behaviours of a domain. The “presentation” abstracts these phenomena and their related concepts in various computer generated forms: visual, acoustic, etc.

9.2.1.1 **Examples**

There are two main examples. One was given in Chapter 3. The other is summarised below. It is from Chapter 8 on “deriving requirements prescriptions from domain descriptions”. The summary follows.

The domain description of Sect. 8.2 outlines an abstract concept of transport nets (of hubs [street intersections, train stations, harbours, airports] and links [road segments, rail tracks, shipping lanes, air-lanes]), their development, traffic [of vehicles, trains, ships and aircraft], etc. We shall assume such a transport domain description below.

Endurants are, for example, presented as follows: (a) transport nets by two dimensional (2D) road, railway or air traffic maps, (b) hubs and links by highlighting parts of 2D maps and by related photos – and their unique identifiers by labeling hubs and links, (c) routes by highlighting sequences of paths (hubs and links) on a 2D map, (d) buses by photographs and by dots at hubs or on links of a 2D map, and (e) bus timetables by, well, indeed, by showing a 2D bus timetable.

Actions are, for example, presented as follows: (f) The insertion or removal of a hub or a link by showing “instantaneous” triplets of “before”, “during” and “after” animation sequences. (g) The start or end of a bus ride by showing flashing animations of the appearance, respectively the flashing disappearance of a bus (dot) at the origin, respectively the destination bus stops.
Events are, for example, presented as follows: (h) A mudslide [or fire in a road tunnel, or collapse of a bridge] along a (road) link by showing an animation of part of a (road) map with an instantaneous sequence of (α) the present link, (β) a gap somewhere on the link, (γ) and the appearance of two (“symbolic”) hubs “on either side of the gap”. (i) The congestion of road traffic “grinding to a halt” at, for example, a hub, by showing an animation of part of a (road) map with an instantaneous sequence of the massive accumulation of vehicle dots moving (instantaneously) from two or more links into a hub.

Behaviours are, for example, presented as follows: (k) A bus tour: from its start, on time, or “thereabouts”, from its bus stop of origin, via (all) intermediate stops, with or without delays or advances in times of arrivals and departures, to the bus stop of destination (ℓ) The composite behaviour of “all bus tours”, meeting or missing connection times, with sporadic delays, with cancellation of some bus tours, etc. – by showing the sequence of states of all the buses on the net.

We say that behaviours ((j)–(ℓ)) are script-based in that they (try to) satisfy a bus timetable ((e)).

9.2.1.2 Towards a Theory of Visualisation and Acoustic Manifestation

The above examples shall serve to highlight the general problem of visualisation and acoustic manifestation. Just as we need sciences of visualising scientific data and of diagrammatic logics, so we need more serious studies of visualisation and acoustic manifestation — so amply, but, this author thinks, inconsistently demonstrated by current uses of interactive computing media.

9.2.2 Simulations

“Simulation is the imitation of some real thing, state of affairs, or process; the act of simulating something generally entails representing certain key characteristics or behaviours of a selected physical or abstract system” [Wikipedia] for the purposes of testing some hypotheses usually stated in terms of the model being simulated and pairs of statistical data and expected outcomes.

9.2.2.1 Explication of Figure 9.1

Figure 9.1 on the next page attempts to indicate four things: (i) Left top: the rounded edge rectangle labeled “The Domain” alludes to some specific domain (“out there”). (ii) Left middle: the small rounded rectangle labeled “A Domain Description” alludes to some document which narrates and formalises a description of “the domain”. (iii) Left bottom: the medium sized rectangle labeled “A Domain Demo based on the Domain Description” (for short “Demo”) alludes to a software system that, in some sense (to be made clear later) “simulates” “The Domain.” (iv) Right: the large rectangle (a) shows a horizontal time axis which basically “divides” that large rectangle into two parts: (b) Above the time axis the “fat” rounded edge rectangle alludes to the time-wise behaviour, a domain trace, of “The Domain” (i.e., the actual, the real, domain). (c) Below the time axis there are eight “thin” rectangles. These are labels S1, S2, S3, S4, S5, S6, S7 and S8. (d) Each of these denote a “run”, i.e., a time-stamped “execution”, a program trace, of the “Demo”. Their “relationship” to the time axis is this: their execution takes place in the real time as related to that of “The Domain” behaviour.

A trace (whether a domain or a program execution trace) is a time-stamped sequence of states: domain states, respectively demo, simulator, monitor and monitor & control states.

From Fig. 9.1 on the following page and the above explication we can conclude that “executions” S4 and S5 each share exactly one time point, t, at which “The Domain” and “The Simulation” “share” time, that is, the time-stamped execution S4 and S5 reflect a “Simulation” state which at time t should reflect (some abstraction of) “The Domain” state.

Only if the domain behaviour (i.e., trace) fully “surrounds” that of the simulation trace, or, vice-versa (cf. Fig. 9.1|S4,S5|), is there a “shared” time. Only if the ‘begin’ and ‘end’ times of the domain behaviour are identical to the ‘start’ and ‘finish’ times of the simulation trace, is there an infinity of shared 1–1 times. Only then do we speak of a real-time simulation.

In Fig 9.2 on Page 249 we show “the same” “Domain Behaviour” (three times) and a (1) simulation, a (2) monitoring and a (3) monitoring & control, all of whose ‘begin/start’ (b/β) and ‘end/finish’ (e/ε) times
coincide. In such cases the “Demo/Simulation” takes place in real-time throughout the ‘begin· · · end’ interval.

Let $\beta$ and $\varepsilon$ be the ‘start’ and ‘finish’ times of either $S_4$ or $S_5$. Then the relationship between $t$, $\beta$, $\varepsilon$, $b$ and $e$ is $t - b = \frac{\beta - e}{2}$ — which leads to a second degree polynomial in $t$ which can then be solved in the usual, high school manner.

### 9.2.2.2 Script-based Simulation

A script-based simulation is the behaviour, i.e., an execution, of, basically, a demo which, step-by-step, follows a script: that is a prescription for highlighting endurants, actions, events and behaviours.

Script-based simulations where the script embodies a notion of time, like a bus timetable, and unlike a route, can be thought of as the execution of a demos where “chunks” of demo operations take place in accordance with “chunks” of script prescriptions. The latter (i.e., the script prescriptions) can be said to represent simulated (i.e., domain) time in contrast to “actual computer” time. The actual times in which the script-based simulation takes place relate to domain times as shown in Simulations $S_1$ to $S_8$ in Fig. 9.1 and in Fig. 9.2(1–3). Traces Fig. 9.2(1–3) and $S_8$ Fig. 9.1 are said to be real-time: there is a one-to-one mapping between computer time and domain time. $S_1$ and $S_4$ Fig. 9.1 are said to be macroscopic: disjoint computer time intervals map into distinct domain times. $S_2$, $S_3$, $S_5$, $S_6$ and $S_7$ are said to be macroscopic: disjoint domain time intervals map into distinct computer times.

In order to concretise the above “vague” statements let us take the example of simulating bus traffic as based on a bus timetable script. A simulation scenario could be as follows. Initially, not relating to any domain time, the simulation “demos” a net, available buses and a bus timetable. The person(s) who are requesting the simulation are asked to decide on the ratio of the domain time interval to simulation time interval. If the ratio is 1 a real-time simulation has been requested. If the ratio is less than 1 a microscopic simulation has been requested. If the ratio is larger than 1 a microscopic simulation has been requested. A chosen ratio of, say 48 to 1 means that a 24 hour bus traffic is to be simulated in 30 minutes of elapsed simulation time. Then the person(s) who are requesting the simulation are asked to decide on the “sampling times” or “time intervals”: If ‘sampling times’ are chosen, then the simulation is stopped at corresponding simulation times: 0 sec., 37.5 sec., 75 sec., 150 sec., 225 sec., 262.5 sec. and 300 sec. The simulation then shows the state of selected endurants and actions at these domain times. If ‘sampling time interval’ is chosen and is set to every 5 min., then the

---

3 We deliberately leave the notion of chunk vague so as to allow as wide an spectrum of simulations.
9.2 Interpretations

The simulation shows the state of selected endurants and actions at corresponding domain times. The simulation is resumed when the person(s) who are requesting the simulation so indicates, say by a “resume” icon click. The time interval between adjacent simulation stops and resumptions contribute with 0 time to elapsed simulation time – which in this case was set to 5 minutes. Finally the requestor provides some statistical data such as numbers of potential and actual bus passengers, etc.

Then two clocks are started: a domain time clock and a simulation time clock. The simulation proceeds as driven by, in this case, the bus time table. To include “unforeseen” events, such as the wreckage of a bus (which is then unable to complete a bus tour), we allow any number of such events to be randomly scheduled. Actually scheduled events “interrupts” the “programmed” simulation and leads to thus unscheduled stops (and resumptions) where the unscheduled stop now focuses on showing the event.

9.2.2.3 The Development Arrow

The arrow, \( \Rightarrow \), between a pair of boxes (of Fig. 9.1 on the facing page) denote a step of development: (i) from the domain box to the domain description box, \( \Rightarrow \), it denotes the development of a domain description based on studies and analyses of the domain; (ii) from the domain description box to the domain demo box, \( \Rightarrow \), it denotes the development of a software system — where that development assumes an intermediate requirements box which has not been shown; (iii) from the domain demo box to either of a simulation traces, \( \Rightarrow \), it denotes the development of a simulator as the related demo software system, again depending on whichever special requirements have been put to the simulator.

9.2.3 Monitoring & Control

Figure 9.2 shows three different kinds of uses of software systems where (2) [Monitoring] and (3) [Monitoring & Control] represent further developments from the demo or simulation software system mentioned in Sect. 9.2.1 and Sect. 9.2.2.2 on the facing page. We have added some (three) horizontal and

![Diagram](image)

Legend: \( mi, mj, ..., mk \): monitorings; \( cx, ..., cy \): controls

Fig. 9.2. Simulation, Monitoring and Monitoring & Control

labeled (p, q and r) lines to Fig. 9.2(1,2,3) (with respect to the traces of Fig. 9.1 on the facing page). They each denote a trace of a endurant, an action or an event, that is, they are traces of values of these phenomena or concepts. A (named) endurant value entails a description of the endurant, whither atomic (‘hub’, ‘link’, ‘bus timetable’) or composite (‘net’, ‘set of hubs’, etc.): of its unique identity, its mereology and a selection
of its attributes. A (named) action value could, for example, be the pair of the before and after states of the action and some description of the function (‘insertion of a link’, ‘start of a bus tour’) involved in the action. A (named) event value could, for example, be a pair of the before and after states of the endurants causing, respectively being effected by the event and some description of the predicate (‘mudslide’, ‘break-down of a bus’) involved in the event. A cross section, such as designated by the vertical lines (one for the domain trace, one for the “corresponding” program trace) of Fig. 9.2 on the previous page(1) denotes a state: a domain, respectively a program state.

Figure 9.2(1) attempts to show a real-time demo or simulation for the chosen domain. Figure 9.2(2) purports to show the deployment of real-time software for monitoring (chosen aspects of) the chosen domain. Figure 9.2(3) purports to show the deployment of real-time software for monitoring as well as controlling (chosen aspects of) the chosen domain.

9.2.3.1 Monitoring

By domain monitoring we mean “to be aware of the state of a domain”, its endurants, actions, events and behaviour. Domain monitoring is thus a process, typically within a distributed system for collecting and storing state data. In this process “observation” points — i.e., endurants, actions and where events may occur — are identified in the domain, cf. points p, q and r of Fig. 9.2. Sensors are inserted at these points. The “downward” pointing vertical arrows of Figs. 9.2(2–3), from “the domain behaviour” to the “monitoring” and the “monitoring & control” traces express communication of what has been sensed (measured, photographed, etc.) as directed by and as input data (etc.) to these monitors. The monitor (being “executed”) may store these “sensings” for future analysis.

9.2.3.2 Control

By domain control we mean “the ability to change the value” of endurants and the course of actions and hence behaviours, including prevention of events of the domain. Domain control is thus based on domain monitoring. Actuators are inserted in the domain “at or near” monitoring points or at points related to these, viz. points p and r of Fig. 9.2 on the preceding page(3). The “upward” pointing vertical arrows of Fig. 9.2 on the previous page(3), from the “monitoring & control” traces to the “domain behaviour” express communication, to the domain, of what has been computed by the controller as a proper control reaction in response to the monitoring.

9.2.4 Machine Development

9.2.4.1 Machines

By a machine we shall understand a combination of hardware and software. For demos and simulators the machine is “mostly” software with the hardware typically being graphic display units with tactile instruments. For monitors the “main” machine, besides the hardware and software of demos and simulators, additionally includes sensors distributed throughout the domain and the technological machine means of communicating monitored signals from the sensors to the “main” machine and the processing of these signals by the main machine. For monitors & controllers the machine, besides the monitor machine, further includes actuators placed in the domain and the machine means of computing and communicating control signals to the actuators.

9.2.4.2 Requirements Development

Essential parts of Requirements to a Machine can be systematically “derived” from a Domain description. These essential parts are the domain requirements and the interface requirements. Domain requirements are those requirements which can be expressed, say in narrative form, by mentioning technical terms only of the domain. These technical terms cover only phenomena and concepts (endurants, actions, events and
behaviours) of the domain. Some domain requirements are projected, instantiated, made more deterministic and extended. We bring examples that are taken from Sect. 8.2, cf. Sect. 9.2.1.1 on Page 246 of the present chapter. (a) By domain projection we mean a sub-setting of the domain description; parts are left out which the requirements stake-holders, collaborating with the requirements engineer, decide is of no relevance to the requirements. For our example it could be that our domain description had contained models of road net attributes such as “the wear & tear” of road surfaces, the length of links, states of hubs and links (that is, [dis]allowable directions of traffic through hubs and along links), etc. Projection might then omit these attributes. (b) By domain instantiation we mean a specialisation of endurants, actions, events and behaviours, refining them from abstract simple entities to more concrete such, etc. For our example it could be that we only model freeways or only model road-pricing nets – or any one or more other aspects. (c) By domain determination we mean that of making the domain description cum domain requirements prescription less non-deterministic, i.e., more deterministic (or even the other way around!). For our example it could be that we had domain-described states of street intersections as not controlled by traffic signals – where the determination is now that of introducing an abstract notion of traffic signals which allow only certain states (of red, yellow and green). (d) By domain extension we basically mean that of extending the domain with phenomena and concepts that were not feasible without information technology. For our examples we could extend the domain with bus mounted GPS gadgets that record and communicate (to, say a central bus traffic computer) the more-or-less exact positions of buses – thereby enabling the observation of bus traffic. Interface requirements are those requirements which can be expressed, say in narrative form, by mentioning technical terms both of the domain and of the machine. These technical terms thus cover shared phenomena and concepts, that is, phenomena and concepts of the domain which are, in some sense, also (to be) represented by the machine. Interface requirements represent (i) the initialisation and “on-the-fly” update of machine endurants on the basis of shared domain endurants; (ii) the interaction between the machine and the domain while the machine is carrying out a (previous domain) action; (iii) machine responses, if any, to domain events — or domain responses, if any, to machine events cum “outputs”; and (iv) machine monitoring and machine control of domain phenomena. Each of these four (i–iv) interface requirement facets themselves involve projection, instantiation, determination, extension and fitting. Machine requirements are those requirements which can be expressed, say in narrative form, by mentioning technical terms only of the machine. (An example is: visual display units.)

9.2.5 Verifiable Software Development

9.2.5.1 An Example Set of Conjectures

We illustrate some conjectures.

(A) From a domain, \( D \), one can develop a domain description \( D \), \( D \) cannot be [formally] verified. It can be [informally] validated “against” \( D \). Individual properties, \( P_D \), of the domain description \( D \) and hence, purportedly, of the domain, \( D \), can be expressed and possibly proved \( D \models P_D \) and these may be validated to be properties of \( D \) by observations in (or of) that domain.

(B) From a domain description, \( D \), one can develop requirements, \( R_{DE} \), for, and from \( R_{DE} \) one can develop a domain demo machine specification \( M_{DE} \) such that \( D, M_{DE} \models R_{DE} \). The formula \( D, M \models R \) can be read as follows: in order to prove that the Machine satisfies the Requirements, assumptions about the Domain must often be made explicit in steps of the proof.

(C) From a domain description, \( D \), and a domain demo machine specification, \( S_{DE} \), one can develop requirements, \( R_{SI} \), for, and from such a \( R_{SI} \) one can develop a domain simulator machine specification \( M_{SI} \) such that \( (D, M_{DE}), M_{SI} \models R_{SI} \). We have “lumped” \( (D, M_{DE}) \) as the two constitute the extended domain for which we, in this case of development, suggest the next stage requirements and machine development to take place.

(D) From a domain description, \( D \), and a domain simulator machine specification, \( M_{SI} \), one can develop requirements, \( R_{MO} \), for, and from such a \( R_{MO} \) one can develop a domain monitor machine specification \( M_{MO} \) such that \( (D, M_{SI}), M_{MO} \models R_{MO} \).

\(^4\) We omit consideration of fitting.
(E) From a domain description, \( D \), and a domain monitor machine specification, \( M_{MO} \), one can develop requirements, \( R_{MC} \), for, and from such a \( R_{MC} \) one can develop a domain monitor & controller machine specification \( M_{MC} \) such that \( (D;M_{MO}), M_{MC} \models R_{MC} \).

### 9.2.5.2 Chains of Verifiable Developments

The above illustrated just one chain (A–E) of developments. There are others. All are shown in Fig. 9.3.

---

**Legend:** D domain, R requirements, M machine

\[ \begin{align*}
\text{DE: Demo, SI: Simulator, MO: Monitor, MC: Monitor & Controller} \\
\text{Fig. 9.3. Chains of Verifiable Developments}
\end{align*} \]

Figure 9.3 can also be interpreted as prescribing a widest possible range of machine cum software products [105, 317] for a given domain. One domain may give rise to many different kinds of Demo machines, Simulators, Monitors and Monitor & Controllers (the unprimed versions of the \( M_T \) machines (where \( T \) ranges over DE, SI, MO, MC)). For each of these there are similarly, “exponentially” many variants of successor machines (the primed versions of the \( M_T \) machines). What does it mean that a machine is a primed version? Well, here it means, for example, that \( M'_{SI} \) embodies facets of the demo machine \( M_{DE} \), and that \( M''_{MC} \) embodies facets of the demo machine \( M_{DE} \), of the simulator \( M'_{SI} \), and the monitor \( M''_{MO} \). Whether such requirements are desirable is left to product customers and their software providers [105, 317] to decide.

### 9.3 Summary

Our divertimento is almost over. It is time to conclude.

#### 9.3.1 What Have We Achieved

We have characterised a spectrum of strongly domain-related as well as strongly inter-related (cf. Fig. 9.3) software product families: demos, simulators, monitors and monitor & controllers. We have indicated varieties of these: simulators based on demos, monitors based on simulators, monitor & controllers based on monitors, in fact any of the latter ones in the software product family list as based on any of the earlier ones. We have sketched temporal relations between simulation traces and domain behaviours: \( a \ priori, a \ posteriori, macroscopic \) and \( microscopic \), and we have identified the real-time cases which lead on to monitors and monitor & controllers.
9.3.2 What Have We Not Achieved — Some Conjectures

We have not characterised the software product family relations other than by the $\mathcal{D}, \mathcal{M} \models \mathcal{R}$ and $(\mathcal{D}; \mathcal{M}_{XYZ}), \mathcal{M} \models \mathcal{R}$ clauses. That is, we should like to prove conjectured type theoretic inclusion relations like:

$$\varphi([\llbracket \mathcal{M}_{X \text{mod ext.}} \rrbracket]) \supseteq \varphi([\llbracket \mathcal{M}_{X' \text{mod ext.}} \rrbracket]), \quad \varphi([\llbracket \mathcal{M}_{X'' \text{mod ext.}} \rrbracket]) \supseteq \varphi([\llbracket \mathcal{M}_{X' \text{mod ext.}} \rrbracket])$$

where $X$ and $Y$ range appropriately, where $\llbracket \mathcal{M} \rrbracket$ expresses the meaning of $\mathcal{M}$, where $\varphi([\llbracket \mathcal{M} \rrbracket])$ denote the space of all machine meanings and where $\varphi([\llbracket \mathcal{M}_{X \text{mod ext.}} \rrbracket])$ is intended to denote that space modulo (“free of”) the $y$ facet (here ext., for extension).

That is, it is conjectured that the set of more specialised, i.e., $n$ primed, machines of kind $x$ is type theoretically “contained” in the set of $m$ primed (unprimed) $x$ machines ($0 \leq m < n$).

There are undoubtedly many such interesting relations between the DEMO, SIMULATOR, MONITOR and MONITOR & CONTROLLER machines, unprimed and primed.

9.3.3 What Should We Do Next?

This chapter has the subtitle: *A Divertimento of Ideas and Suggestions*. It is not a proper theoretical chapter. It tries to throw some light on families and varieties of software, i.e., their relations. It focuses, in particular, on so-called DEMO, SIMULATOR, MONITOR and MONITOR & CONTROLLER software and their relation to the “originating” domain, i.e., that in which such software is to serve, and hence that which is being extended by such software, cf. the compounded ‘domain’ $(\mathcal{D}; \mathcal{M}_i)$ of in $(\mathcal{D}; \mathcal{M}_i), \mathcal{M}_j \models \mathcal{D}$. These notions should be studied formally. All of these notions: requirements projection, instantiation, determination and extension can be formalised. The specification language, in the form used here (without CSP processes, [238]) has a formal semantics and a proof system So the various notions of development, $(\mathcal{D}; \mathcal{M}_i), \mathcal{M}_j \models \mathcal{R}$ and $\varphi(\mathcal{M})$ can be formalised.
We briefly (i) summarise key concepts of the intrinsics of domain analysis & description, domain facets, requirements engineering; (ii) put forward a number of diverse observations; (iii) and finally quote Tony Hoare’s observations on domain engineering.

10.1 Programming Languages and Domains

In the quest for precise understanding of programming languages one studies formal syntax and formal semantics for and of these. A programming language is a ‘language’ “spoken” by and “written in” by programmers. The domain analysis & description that we have here pursued has been done so in a quest to understand the language spoken amongst professionals of the domain being modelled. In that sense the two endeavours ‘parallel’. The IBM Vienna Labor, from around the mid 1960s to a little beyond the mid 1970s, was a unique center for the study and practice of programming language semantics. One early achievement is reflected in the first conference on Formal Language Description Languages [350]. Subsequent achievements were [23, 24, 25, 22]. This author was only there for a short period of two years, 1973–1975. They have determined, to this day, my professional, scientific and engineering focus and direction. I am deeply indebted to colleagues such as the late Peter Lucas, to Kurt Walk, and to Cliff Jones.

10.2 Summary of Chapters 3–6

We refer to the main, the full endurant/perdurant ontology, diagram, Fig. 3.1 on Page 44.

Now re-explain what is going on, method-wise, with respect to that diagram, including how the internal qualities glue it all together.

10.2.1 Chapter 3: External Qualities

The left side of the diagram, labeled external qualities, takes up a sizable area of the whole. It designates our tackling the analysis of the external qualities of endurants first. The aim of that analysis is to uncover the entire collection of all observable endurants of a domain. One does so by asking questions of inspected entities as suggested by the analysis prompts. And when this analysis end up with

- atomic,
- composite,
- part-materials,
- material-parts,
- part-parts conjoins and material

endurants, one applies a description prompt and starts all over again with analysis and description till all endurants have been external quality-analysed and described, and one is ready to analyse & describe internal qualities.
10.2.2 Chapter 4: Internal Qualities

The bottom side of Fig. 3.1 on Page 44, labeled internal qualities, takes up a not so visible area of the whole. But it reflects every bit as an important aspect of domain science & engineering. It designates our tackling the analysis of the internal qualities of endurants in the order unique identifiers, mereologies and attributes in the order, “strictly” (!). The internal qualities is what gives “meaning” to endurants.

10.2.3 Chapter 5: Transcendentalism

Transcendentalism represents a new way of looking at domain description. Before, we claim, there were endurants and perdurants; with no “obvious connection”. Now, we claim, perdurants are transcendentally related to endurants; and strongly so.

10.2.4 Chapter 6: Perdurants

The deduction of behaviour signature and definition elements such as channels from mereologies, values from static attributes, state variables in the form of “update-able” parameters from programmable attributes, and part variables and their access (and possible update) from other dynamic attributes, marks another contribution of domain analysis & description.

10.3 A Final Summary of Triptych Concepts

The “near exhaustive” summary listings that now follow serve the purpose of reminding the reader of the rather large, perhaps even “exhausting” set of new terms, each to be appropriated thoroughly and now applied!

10.3.1 The Intrinsics of Domain Analysis & Description

This is a summary of the calculi of Chapters 3–6. There are the following issues to be dealt with in an analysis & description of the intrinsics of domain analysis & description, and in the transcendental deduction of parts into behaviours.

- **External qualities:**
  - **The Analysis Prompts:**
    - Entities
    - Endurants and Perdurants
    - Endurants: Discrete and Continuous
    - Discrete Endurants: Physical Parts, Structures and Living Species
    - Physical Parts: Natural Parts and Artefacts
    - Physical Parts: Structures
    - Physical Parts: Living Species – Plants and Animals
    - Continuous Endurants: Materials
    - Natural Parts and Artefacts: Atomic, Composite, Concrete and Conjoins
  - **The Description Prompts:**
    - Endurant Observers

- **Internal Qualities:**
  - Unique Identifiers
  - Mereology
  - Attributes
  - Intentionality

- **Transcendental Deduction**
10.4 Systems Development

We see computing systems development as comprising the development of hardware and software. For software that comprises the development of, or reliance on existing, appropriate domain descriptions; the development of requirements prescriptions based on these domain models – including but not shown in this monograph, the analysis and prescription of unchanged or changed business processes also known as business process re-engineering [198, 197, 244, 250]. In the context of formal requirements development we refer to [41, Sect. 19.3: Business Process Reengineering Requirements]. There is a whole new dimension, we claim, to business process engineering and re-engineering (BPE&BPR) in the light of domain analysis & description. Yes, we suggest that someone reviews the possible foundation for BPE&BPR.
10.5 On How to Conduct a Domain Analysis & Description Project

We have established a scientific & engineering discipline of domain analysis & description, part II, and of domain requirements, Chapter 8. We have shown, in the very many examples of this monograph and in quite a collection of experimental studies [80], that that discipline can be applied; but can it be applied just because the reader has now studied that discipline? What is not covered in this monograph is the practical aspects of carrying out the “theory & practice” of constructing domain models. We shall, in itemized form, suggest an approach that we have applied for over 50 years, since the 1973–1975 PL/I compiler project at the IBM Vienna Laboratory; in the CHILL [193] and Ada [91, 92, 127, 300] compiler development projects at The Dansk Datamatik Center, DDC in the 1980s and more.

• It is assumed that you have a team of, for example, 5–7 professional software engineers, persons well-versed¹ in the concepts and method of for example [39, 40, 41] as well, now, of this monograph.
• First you set aside a month-long, preliminary study of the domain at hand: not a study where you neither analyse nor describe the domain, just a simple literature study, using, for example also the Internet.
• Then you conduct, say over a three month period, an experimental domain analysis & description.
  ∞ One purpose of this experiment is to test whether an assumed set of analysis & description prompts “will do the trick”, or whether the project must revise the upper ontology for that domain.
  ∞ Another purpose is to structure the project group. During the experiment some first thoughts on major endurants should emerge – and the project group structured accordingly: one project member per major, often composite, part to be responsible for all aspects of the analysis & description of that part – and its composites. On a rotating shift basis other project members shall act as reviewers of each others’ work.
• Then the project can enter its application stage.²
  ∞ The external qualities step of the domain analysis & description mainly consists of the Endurant Observers, Sect. 3.18 step. It is the first serious step. It is to be followed by the next steps in strict order.
    ∞ The unique identifier step, Sect. 4.2, in which unique identifiers for all relevant endurant categories are settled.
    ∞ The mereology step, Sect. 4.3, in which the mereology for all relevant endurant categories are settled. This is a crucial step. Care must be taken. This step requires intensive interaction between project members. We advice that each project member “play around” with mereology invariants.
    ∞ The attributes step, Sect. 4.4, in which attributes for all relevant endurant categories are settled. This step is less interaction-intensive – although those attributes which shall later form the basis for work on intentionalities do require some interaction.
    ∞ The intentionality step, Sect. 4.5, has an as yet not fully understood element of engineering research. It completes the first iteration of work on internal qualities.
  ∞ The first iteration of work on both external and internal qualities will usually be followed by several further such iterations – in between the next steps.
  ∞ These next steps are those of transcendental deductive work on perdurants – also to be pursued in “strict order”.
    ∞ In the states step, see Sect. 6.2, we value define states of all invariants, … .
    ∞ In the channels and communication, see Sect. 6.5,
    ∞ In the perdurant signatures, Sect. 6.6,
    ∞ In the discrete behaviour definitions, see Sect. 6.3.4,
    ∞ In the discrete action behaviour definitions, see Sect. 6.10,
    ∞ In the discrete event behaviour definitions, see Sect. 6.11,

¹ That is: they have a reasonably qualified knowledge of this monograph, can apply this knowledge, have a reasonably qualified knowledge of discrete mathematics, of mathematical logic for computing scientists, formal methods, functional, logic, imperative and parallel programming, and posses both analytic skills and master their mother tongue and English, if it is not their mother tongue.
² We refer to this study, experiment, apply triplet as SEA.
10.8 Tony Hoare’s Reaction to ‘Domain Modelling’

We close this monograph as we opened it: As the first item of this monograph Item 1 on Page 3, we quoted Tony Hoare. It is likewise fitting to bring as final text also a quote from his hand. In a 2006 e-mail, in response, undoubtedly to my steadfast – perhaps conceived as stubborn – insistence, on domain engineering, Tony Hoare summed up his reaction to domain engineering as follows, and I quote:\footnote{E-Mail to Dines Bjørner, July 19, 2006}:

\textit{'There are many unique contributions that can be made by domain modelling.'}
• The models describe all aspects of the real world that are relevant for any good software design in the area. They describe possible places to define the system boundary for any particular project.
• They make explicit the preconditions about the real world that have to be made in any embedded software design, especially one that is going to be formally proved.
• They describe the whole range of possible designs for the software and the whole range of technologies available for its realisation.
• They provide a framework for a full analysis of requirements, which is wholly independent of the technology of implementation.
• They enumerate and analyse the decisions that must be taken earlier or later in any design project, and identify those that are independent and those that conflict. Late discovery of feature interactions can be avoided.”

Whether they will be made — these contributions — is up to reader!
11

BIBLIOGRAPHY

11.1 Bibliographical Notes

TO BE WRITTEN

11.2 References

10. K. Araki et al., editors. IFM 1999–2018: Integrated Formal Methods,
    and
    - 2nd IFM, 2000: LNCS 1945,
    - 3rd IFM, 2002: LNCS 2335,
    - 4th IFM, 2004: LNCS 2999,
    - 5th IFM, 2005: LNCS 3771,
    - 6th IFM, 2007: LNCS 4591,
    - 7th IFM, 2009: LNCS 5423,
    - 8th IFM, 2010: LNCS 6396,
    - 9th IFM, 2012: LNCS 7321,
    - 10th IFM, 2013: LNCS 7940,
    - 11th IFM, 2014: LNCS 8739,
    - 12th IFM, 2016: LNCS 9681,
    - 13th IFM, 2017: LNCS 10510,
    - 14th IFM, 2018: LNCS 11023,
    Springer
    Lecture Notes in Computer Science


80. Dines Bjørner. Domain Case Studies:
- 2019: Container Line, ECNU, Shanghai, China [URL: imm.dtu.dk/~db/container-paper.pdf]
- 2012: Credit Cards, Uppsala [URL: imm.dtu.dk/~dibj/2016/credit/accs.pdf]
- 2010: Web-based Transaction Processing, Techn. Univ. of Vienna, Austria [URL: imm.dtu.dk/~dibj/wfdftp.pdf]

Experimental research reports, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark.


114. Roberto Casati and Achille C. Varzi, editors. Events. Ashgate Publishing Group – Dartmouth Publishing Co. Ltd., Wey Court East, Union Road, Farnham, Surrey, GU9 7PT, United Kingdom, 23 March 1996.


126. Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí Oliet, José Meseguer, and Carolyn Talcott. Maud 2.6 Manual. Department of Computer Science, University of Illinois and Urbana-Champaign, Ill., USA, January 2011.


11.2 References


[URL: cl.cam.ac.uk/techreports/UCAM-CL-TR-391.pdf].


283. Theodore McCombs. Maude 2.0 Primer. Department of Computer Science, University of Illinois and Urbana-Champaign, Urbana-Champaign, Ill., USA, August 2003.


1 holey: something full of holes
APPENDICES
Appendix: A PIPELINES DOMAIN – ENDURANTS

In this appendix we present an example description of the endurants of a domain of pipelines. Thus the example illustrates major aspects of a domain of con joins. The various sections slavishly follow the steps of domain analyser & describer: endurants, Sect. A.1 unique identifiers, Sect. A.2 mereologies, Sect. A.3 attributes. Sect. A.4

A.1 Parts and Material

A.1.1 Flow Net Parts

The concept of a flow net\(^1\) is illustrated in, for example, oil pipelines. See Figure A.1.

Fig. A.1. Top row: oil unit graphics; diagram of a simple oil pipeline. Bottom row: a pump; a valve; map of the Trans-Alaska Pipeline System (TAPS); photo of TAPS.

A.1.1.1 Narrative

483 There is the pipeline system pl:PL.
484 From a pipeline system we choose to observe a pipeline aggregate of conjoined pipe elements, pla:PLA.

\(^{1}\) – of conjoined parts and materials
The pipeline aggregate of conjoined pipe elements is modelled here as a set of conjoined pipe elements, \(\text{cps}:\text{CPs}\).

By a conjoined pipe element, \(\text{pe}:\text{PE}\), we here mean the conjoined pipe element, i.e., \(\text{pe}:\text{PE}\), from which we choose to observe one material, \(\text{m}:\text{M}\), here the oil.

A conjoined pipe element\(^2\), \(\text{pe}:\text{PE}\), is either a well (a volume from which material is pumped), a pump (which moves the fluids, \(\text{m}:\text{M}\), by mechanical action), a pipe (along which material can move), a valve (which is either fully open, or fully closed, or at some position in-between thus facilitating to a full degree or some partial degree, or hinder the flow of, in this case, oil), a fork (which “diverts” [in this example] a single flow into two flows), a join (which “merges” [in this example] two flows into a single flow), or a sink (a volume into which material is “spilled”).

### A.1.1.2 Formalisation

<table>
<thead>
<tr>
<th>type</th>
<th>491. Va :: Valve</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL</td>
<td>492. Fo :: Fork</td>
</tr>
<tr>
<td>PLA</td>
<td>493. Jo :: Join</td>
</tr>
<tr>
<td>cPEs = PE-set</td>
<td>494. Si :: Sink</td>
</tr>
<tr>
<td>PE == We</td>
<td>Pu</td>
</tr>
<tr>
<td>M</td>
<td>484. obs_PLA: PL \rightarrow PLA</td>
</tr>
<tr>
<td>We :: Well</td>
<td>485. obs_cPEs: PLA \rightarrow cPEs</td>
</tr>
<tr>
<td>Pu :: Pump</td>
<td>487. obs_M: PE \rightarrow M</td>
</tr>
<tr>
<td>Pi :: Pipe</td>
<td></td>
</tr>
</tbody>
</table>

### A.1.2 Pipeline States

#### A.1.2.1 Narrative

Given a pipeline, \(\text{pl}:\text{PL}\), we can calculate the set of all its pipe elements.

#### A.1.2.2 Formalisation

<table>
<thead>
<tr>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_pipeline_units: PL \rightarrow PE-set</td>
</tr>
<tr>
<td>all_pipeline_units(pl) \equiv obs_cPEs(obs_PLS(pl))</td>
</tr>
</tbody>
</table>

### A.2 Unique Identifiers

There is a set of identifiers, \(\text{UI}\).

Each pipe unit is endowed with such an identifier.

All such identifiers of pipe elements of a pipeline are distinct, i.e., unique: no two pipe elements are endowed with identical such identifiers.

\(^2\) We ignore join-fork and redirect units.
\begin{verbatim}
A.3 Mereologies

A.3.1 The Pipeline Unit Mereology

Pipeline units serve to conduct fluid or gaseous material. The flow of these occur in only one direction: from so-called input to so-called output.

499 Wells have exactly one connection to an output unit.
500 Pipes, pumps, valves and redirectors have exactly one connection from an input unit and one connection to an output unit.
501 Forks have exactly one connection from an input unit and exactly two connections to distinct output units.
502 Joins have exactly two connections from distinct input units and one connection to an output unit.
503 Sinks have exactly one connection from an input unit.

Thus we model the mereology of a pipeline unit as a pair of disjoint sets of unique pipeline unit identifiers.

\begin{verbatim}
504 \textit{type} PM' = (\textit{UI-set} \times \textit{UI-set})
504 \textit{value} \{ (iuis,ouis) \mid iuis \cap ouis = \{ \} \}
\end{verbatim}

A.3.2 Partial Wellformedness of Pipelines, 0

The well-formedness inherent in narrative lines 499–503 are formalised:

\begin{verbatim}
axiom \textit{Well-formedness of Pipeline Systems, PL (0)}
\forall p\in PL, p\in all_pipeline_units(pl) \Rightarrow
\begin{align*}
\text{let } (iuis,ouis) &= \textit{mereo PE}(pe) \text{ in } \\
\text{case \{ card iuis, card ouis \} of }
\end{align*}
\begin{align*}
499 & (0,1) \rightarrow \textit{is We}(pe), \\
500 & (1,1) \rightarrow \textit{is Pi}(pe) \land \textit{is Pu}(pe) \land \textit{is Va}(pe), \\
501 & (1,2) \rightarrow \textit{is Fo}(pe), \\
502 & (2,1) \rightarrow \textit{is Jo}(pe), \\
503 & (1,0) \rightarrow \textit{is Si}(pe), \quad \text{false}
\end{align*}
\end{verbatim}

A.3.3 Partial Well-formedness of Pipelines, 1

To express full well-formedness we need express that pipeline nets are acyclic. To do so we first define a function which calculates all routes in a net.
\end{verbatim}
A.3.3.1 Shared Connectors

Two pipeline units, \( p_i \) with unique identifier \( \pi_i \), and \( p_j \) with unique identifier \( \pi_j \), that are connected, such that an outlet marked \( \pi_j \) of \( p_i \) “feeds into” inlet marked \( \pi_i \) of \( p_j \), are said to share the connection (modeled by, e.g., \( \{(\pi_i, \pi_j)\} \)).

A.3.3.2 Routes

The observed pipeline units of a pipeline system define a number of routes (or pipelines):

- **Basis Clauses:**
  - The null sequence, \( \langle \rangle \), of no units is a route.
  - Any one pipeline unit, \( p_e \), of a pipeline system forms a route, \( \langle p_e \rangle \), of length one.

- **Inductive Clauses:**
  - Let \( r_i \hat\langle p_i \rangle \) and \( \langle p_j \rangle \hat r_j \) be two routes of a pipeline system.
  - Let \( p_e_i \) and \( p_e_j \) be the unique identifiers \( p_i \), respectively \( p_j \).
  - If one of the output connectors of \( p_e_i \) is \( p_e_i \), and one of the input connectors of \( p_e_j \) is \( p_e_j \),
    then \( r_i \hat\langle p_i \rangle \hat\langle p_j \rangle \hat r_j \) is a route of the pipeline system.

- **Extremal Clause:**
  - Only such routes which can be formed by a finite number of applications of the clauses form a route.

\[
\begin{align*}
R &= \text{PE}^\omega \\
\text{value} \quad \text{routes}(\text{PL} \sim R) &
\end{align*}
\]

A.3.3.3 Wellformed Routes

The observed pipeline units of a pipeline system forms a net subject to the following constraints:

- a unit output connectors, if any, are connected to unit input connectors;
- b unit input connectors, if any, are connected to unit output connectors;
- c there are no cyclic routes;
- d nets has all their connectors connected, that is, “starts” with wells
- e and “ends” with sinks.

\[
\begin{align*}
\text{value} \quad \text{wf}_\text{Net}(\text{PL} \rightarrow \text{Bool}) &
\end{align*}
\]
A.4 Attributes

We speak of four kinds of attributes: Geometric Unit Attributes, Spatial Unit Attributes, Unit Action Attributes and Flow Attributes.

A.4.1 Geometric Unit Attributes

515 Common static unit attributes are Diameters and Lengths.
516 Well units have one output “Diameter”; pipe, Valve, Pump and Redirector units have Diameter; and Sink units have one input “Diameter”.
517 Pipe, valve and pumps units have Length.
518 Fork units have one input Diameter, two output Diameters: iD, oD1, oD2, and Lengths from input to a fork center, and from that to the two outputs: iL, oL1, oL2.
519 Join units have the “reverse”: one output Diameter, two input Diameters: oD, iD1, iD2, and Lengths from the two inputs to a join center, and from that to the single output: iL1, iL2, oL.
520 Redirector units have Lengths from the input to a “center” (where the unit redirection can be said to be “centered”), and from that center to the output: iL, oL.

type
515. D, L

value
516. attr_D: (We|Pi|Va|Pu|Rd|Si) → D
517. attr_L: (Pi|Va|Pu) → L
518. attrDs: Fo → (D×(D×D))
519. attrDs: Fo → (L×(L×L))
519. attrDs: Jo → ((D×D)×D)
519. attrLs: Jo → ((L×L)×L)
520. attrLs: Rd → L×L

We omit detailing the angles with which the two segments emanate from the input segment of fork, the two segments are incident upon the put segment of a join, and a redirector deviates the output segment from its input segment. The oil unit graphics of Fig. A.1 hints at these angles.

A.4.2 Spatial Unit Attributes

Pipelines are laid down in flat and hilly, even mountainous terrains. Any one pipeline unit has spatial locations. We shall refrain from detailing (let alone formalising) the spatial attributes of units. But we can suggest the following: Every unit has some spatial attributes: As material flow in units is one-directional we can associate with any unit a unique point. With pumps, valves, forks and joins we may associate that point with “the middle, center” of the unit. With wells and sinks we may associate that point with the point of the well, respectively the sink, where oil is delivered, respectively accepted from the pipeline. With pipes we suggest to associate that point with the mid-point, “halfway along the pipe”. Similarly we can
associate a **length** with some units. Pumps, valves, forks and joins we suggest to have length 0. So only pipes have lengths. We suggest that the length of a pipe is the actual, perhaps, curved, length between its two end-points. We bring this example as an illustration of the use of analysis and description prompts, and not as an example of a full-fledged pipeline domain description, we shall refrain from systematically narrating and formalising these spatial unit attributes and the consequences of doing so.\(^3\)

### A.4.3 Unit Action Attributes

521 Valve units are either 100% open, or 100% closed.\(^4\)
522 Pump units are either pumping, or not_pumping.\(^5\)

**type**
521. OC == "open" | "closed"
522. PS == "pumping" | "not_pumping"

**value**
521. attr_OC: Va → OC
522. attr_PS: Pu → PS

### A.4.4 Flow Attributes

#### A.4.4.1 Flows and Leaks

We now wish to examine the flow of liquid (or gaseous) material in pipeline units. So we postulate a unit attribute Flow. We use two types

523 **type** Flow, Leak = Flow.

Productive flow, Flow, and wasteful leak, Leak, is measured, for example, in terms of volume of material per second. We then postulate the following unit attributes “measured” at the point of in- or out-flow or in the interior of a unit.

524 current flow of material into a unit input connector,
525 maximum flow of material into a unit input connector while maintaining laminar flow,
526 current flow of material out of a unit output connector,
527 maximum flow of material out of a unit output connector while maintaining laminar flow,
528 current leak of material at a unit input connector,
529 maximum guaranteed leak of material at a unit input connector,
530 current leak of material at a unit input connector,
531 maximum guaranteed leak of material at a unit input connector,
532 current leak of material from “within” a unit, and
533 maximum guaranteed leak of material from “within” a unit.

**type**
523 Flow, Leak = Flow

**value**
524 attr_cur_iFlow: PE → UI → Flow
525 attr_max_iFlow: PE → UI → Flow
526 attr_cur_oFlow: PE → UI → Flow
527 attr_max_oFlow: PE → UI → Flow
528 attr_cur_iLeak: PE → UI → Leak
529 attr_max_iLeak: PE → UI → Leak

\(^3\) The ‘consequences’ alluded to are those of the spatial well-formedness of pipelines.

\(^4\) Without loss of generality we do not model fractional open/closed status.

\(^5\) Without loss of generality we do not model fractional pumping status.
The maximum flow attributes are static attributes and are typically provided by the manufacturer as indicators of flows below which laminar flow can be expected. The current flow attributes may be considered either reactive or biddable attributes.

It may be difficult or costly, or both, to ascertain flows and leaks in materials-based domains. But one can certainly speak of these concepts. This casts new light on domain modeling. That is in contrast to incorporating such notions of flows and leaks in requirements modeling where one has to show implementability. Modeling flows and leaks is important to the modeling of materials-based domains.

For every unit of a pipeline system, except the well and the sink units, the following law apply.

The flows into a unit equal
- the leak at the inputs
- plus the leak within the unit
- plus the flows out of the unit
- plus the leaks at the outputs.

A.4.4.2 Intra Unit Flow and Leak Law

The sum_cur_iFlow (cf. Item 535) sums current input flows over all input connectors.
The sum_cur_iLeak (cf. Item 535a) sums current input leaks over all input connectors.
The sum_cur_oFlow (cf. Item 535c) sums current output flows over all output connectors.
The sum_cur_oLeak (cf. Item 535d) sums current output leaks over all output connectors.

A.4.4.3 Inter Unit Flow and Leak Law

For every pair of connected units of a pipeline system the following law apply:
- the flow out of a unit directed at another unit minus the leak at that output connector.
b equals the flow into that other unit at the connector from the given unit plus the leak at that connector.

\textbf{axiom} [Well-formedness of Pipelines, PL (3)]
\begin{align*}
& \forall pl, pe, pe': PE \cdot \\
& \{ pe, pe' \} \subseteq \text{all pipeline units}(pl) \\
& \land pe \neq pe' \\
& \land \text{let} \ (iuis, ouis) = \text{mereo}_{PE}(pe), \ (iuis', ouis') = \text{mereo}_{PE}(pe') \\
& iui = \text{uid}_{PE}(pe), \ ui' = \text{meruo}_{PE}(pe') \ \text{in} \\
& ui \in iuis \land ui' \in ouis' \Rightarrow \\
& \text{attr}_{\text{curFlow}}(pe')(ui') - \text{attr}_{\text{leakFlow}}(pe')(ui') \\
& = \text{attr}_{\text{curFlow}}(pe)(ui) + \text{attr}_{\text{leakFlow}}(pe)(ui) \\
\end{align*}
\textbf{comment:} b' precedes b

From the above two laws one can prove the \textbf{theorem}: what is pumped from the wells equals what is leaked from the systems plus what is output to the sinks.
MEREOLOGY, A MODEL

We first present informal examples of mereologies. Then an axiom system for mereology. Then a model of mereology. And finally we sketch a proof that the model satisfies the axioms.

B.1 Examples of Illustrating Aspects of Mereology

We present six examples of systems illustrating the concept of mereology.

B.1.1 Air Traffic

Figure B.1 shows nine adjacent (9) boxes and eighteen adjacent (18) lines. Boxes and lines are parts. The line parts “neighbours” the box parts they “connect”. Individually boxes and lines represent adjacent parts of the composite air traffic “whole”. The rounded corner boxes denote buildings. The sharp corner box denote aircraft. Lines denote radio telecommunication. The “overlap” between neighbouring line and box parts are indicated by “connectors”. Connectors are shown as small filled, narrow, either horisontal or vertical “filled” rectangle\(^1\) at both ends of the double-headed-arrows lines, overlapping both the line arrows and the boxes. The index ranges shown attached to, i.e., labeling each unit, shall indicate that there are a multiple of the “single” (thus representative) box or line unit shown. These index annotations are what makes the diagram of Fig. B.1 schematic. Notice that the ‘box’ parts are fixed installations and that the double-headed arrows designate the ether where radio waves may propagate. We could, for example,

\(^1\) There are 36 such rectangles in Fig. B.1.
assume that each such line is characterised by a combination of location and (possibly encrypted) radio communication frequency. That would allow us to consider all lines for not overlapping. And if they were overlapping, then that must have been a decision of the air traffic system.

### B.1.2 Buildings

![Building Plan Diagram](image)

**Fig. B.2.** A building plan with installation

Figure B.2 shows a building plan — as a composite part. The building consists of two buildings, A and H. The buildings A and H are neighbours, i.e., shares a common wall. Building A has rooms B, C, D and E. Building H has rooms I, J and K. Rooms L and M are within K. Rooms F and G are within C. The thick lines labeled N, O, P, Q, R, S, and T models either electric cabling, water supply, air conditioning, or some such “flow” of gases or liquids. Connection $\kappa \iota$ provides means of a connection between an environment, shown by dashed lines, and B or J, i.e. “models”, for example, a door. Connections $\kappa$ provides “access” between neighbouring rooms. Note that “neighbouring” is a transitive relation. Connection $\omega \iota \sigma$ allows electricity (or water, or oil) to be conducted between an environment and a room. Connection $\omega$ allows electricity (or water, or oil) to be conducted through a wall. Et cetera. Thus “the whole” consists of A and H. Immediate sub-parts of A are B, C, D and E. Immediate sub parts of C are G and F. Et cetera.

### B.1.3 A Financial Service Industry

Figure B.3 on the next page is rather rough-sketchy! It shows seven (7) larger boxes [6 of which are shown by dashed lines], six [6] thin lined “distribution” boxes, and twelve (12) double-arrowed lines. Boxes and lines are parts. (We do not described what is meant by “distribution”.) Where double-arrowed lines touch upon (dashed) boxes we have connections. Six (6) of the boxes, the dashed line boxes, are composite parts, five (5) of them consisting of a variable number of atomic parts; five (5) are here shown as having three atomic parts each with bullets “between” them to designate “variability”. Clients, not shown, access the outermost (and hence the “innermost” boxes, but the latter is not shown) through connections, shown by bullets, •.

### B.1.4 Machine Assemblies

Figure B.4 on the facing page shows a machine assembly. Square boxes designate either composite or atomic parts. Black circles or ovals show connections. The full, i.e., the level 0, composite part consists
Examples of Illustrating Aspects of Mereology

The Finance Industry “Watchdog”

The Finance Industry “Watchdog”

The Finance Industry “Watchdog”

Fig. B.3. A Financial Service Industry

Fig. B.4. An air pump, i.e., a physical mechanical system

of four immediate parts and three internal and three external connections. The Pump is an assembly of six (6) immediate parts, five (5) internal connections and three (3) external connectors. Et cetera. Some connections afford “transmission” of electrical power. Other connections convey torque. Two connections convey input air, respectively output air.

B.1.5 Oil Industry

B.1.5.1 “The” Overall Assembly

Figure B.5 on the next page shows a composite part consisting of fourteen (14) composite parts, left-to-right: one oil field, a crude oil pipeline system, two refineries and one, say, gasoline distribution network, two seaports, an ocean (with oil and ethanol tankers and their sea lanes), three (more) seaports, and three, say gasoline and ethanol distribution networks. Between all of the neighbouring composite parts there are connections, and from some of these composite parts there are connections (to an external environment). The crude oil pipeline system composite part will be concretised next.
B.1.5.2 A Concretised Composite Pipeline

Figure B.6 shows a pipeline system. It consists of 32 atomic parts: fifteen (15) pipe units (shown as directed arrows and labeled p1–p15), four (4) input node units (shown as small circles, ○, and labeled inj–inj), four (4) flow pump units (shown as small circles, ○, and labeled fpa–fpd), five (5) valve units (shown as small circles, ○, and labeled vx–vz), three (3) join units (shown as small circles, ○, and labeled jb–jc), two (2) fork units (shown as small circles, ○, and labeled fb–fc), one (1) combined join & fork unit (shown as small circles, ○, and labeled jafa), and four (4) output node units (shown as small circles, ○, and labeled onp–ons).

In this example the routes through the pipeline system start with node units and end with node units, alternates between node units and pipe units, and are connected as shown by fully filled-out dark coloured disc connections. Input and output nodes have input, respectively output connections, one each, and shown as lighter coloured connections. In [60] we present a description of a class of abstracted pipeline systems.

B.1.6 Railway Nets

The left of Fig. B.7 on the next page [L] diagrams four rail units, each with two, three or four connectors shown as narrow, somewhat “longish” rectangles. Multiple instances of these rail units can be assembled...
An Axiom System for Mereology

(i.e., composed) by their connectors as shown on Fig. B.7 [L] into proper rail nets. The right of Fig. B.7

Fig. B.7. Railway Concepts. To the left: Four rail units. To the right: A “model” railway net:

- An Assembly of four Assemblies: two stations and two lines.
- Lines here consist of linear rail units.
- Stations of all the kinds of units shown to the left.
- There are 66 connections and four “dangling” connectors.

[R] diagrams an example of a proper rail net. It is assembled from the kind of units shown in Fig. B.7 [L].

In Fig. B.7 [R] consider just the four dashed boxes: The dashed boxes are assembly units. Two designate stations, two designate lines (tracks) between stations. We refer to the caption four line text of Fig. B.7 for more “statistics”. We could have chosen to show, instead, for each of the four “dangling” connectors, a composition of a connection, a special “end block” rail unit and a connector.

B.1.7 Discussion

We have brought these examples only to indicate the issues of a “whole” and atomic and composite parts, adjacency, within, neighbour and overlap relations, and the ideas of attributes and connections. We shall make the notion of ‘connection’ more precise in the next section.

B.2 An Axiom System for Mereology

Classical axiom systems for mereology focus on just one sort of “things”, namely Parts. Leśniewski had in mind, when setting up his mereology to have it supplant set theory. So parts could be composite and consisting of other, the sub-parts — some of which would be atomic; just as sets could consist of elements which were sets — some of which would be empty.

B.2.1 Parts and Attributes

In our axiom system for mereology we shall avail ourselves of two sorts: Parts, and Attributes.

- type \( \mathcal{P}, \mathcal{A} \)

Attributes are associated with Parts. We do not say very much about attributes: We think of attributes of parts to form possibly empty sets. So we postulate a primitive predicate, \( \epsilon \), relating Parts and Attributes.

- \( \epsilon : \mathcal{A} \times \mathcal{P} \to \text{Bool} \).

Please be open-minded! Do not think of “parts” \( \mathcal{P} \) being “robust” in the sense of being rigid bodies. Think, more of them as point space sets. Of course, parts \( \mathcal{P} \) are really what the below axioms expresses. Allow two or more of these parts to share points, i.e., to “protrude” into one-another; then the axioms are easier, we find, to comprehend.

\[2\] Identifiers \( \mathcal{P} \) and \( \mathcal{A} \) stand for model-oriented types (parts and atomic parts), whereas identifiers \( \mathcal{P} \) and \( \mathcal{A} \) stand for property-oriented types (parts and attributes).
B.2.2 The Axioms

The axiom system to be developed in this section is a variant of that in [115]. We introduce the following relations between parts:

\[ \begin{align*}
\text{part of:} & \quad P : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \quad \text{Page 294} \\
\text{proper part of:} & \quad PP : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \quad \text{Page 294} \\
\text{overlap:} & \quad O : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \quad \text{Page 294} \\
\text{underlap:} & \quad U : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \quad \text{Page 294} \\
\text{over crossing:} & \quad OX : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \quad \text{Page 294} \\
\text{under crossing:} & \quad UX : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \quad \text{Page 295} \\
\text{proper overlap:} & \quad PO : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \quad \text{Page 295} \\
\text{proper underlap:} & \quad PU : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \quad \text{Page 295}
\end{align*} \]

**Part-hood.** $P$, expresses that $p_x$ is a part of $p_y$ as $P(p_x, p_y)$. Part $p_x$ is part of itself (reflexivity) (B.1).

If a part $p_x$ is part of $p_y$ and, vice versa, part $p_y$ is part of $p_x$, then $p_x = p_y$ (anti-symmetry) (B.2). If a part $p_x$ is part of $p_y$ and part $p_y$ is part of $p_z$, then $p_x$ is part of $p_z$ (transitivity) (B.3).

\[ \forall p_x : \mathcal{P} \bullet P(p_x, p_x) \] (B.1)

\[ \forall p_x, p_y : \mathcal{P} \bullet (P(p_x, p_y) \land P(p_y, p_x)) \Rightarrow p_x = p_y \] (B.2)

\[ \forall p_x, p_y, p_z : \mathcal{P} \bullet (P(p_x, p_y) \land P(p_y, p_z)) \Rightarrow P(p_x, p_z) \] (B.3)

**Proper Part-hood.** $PP$, expresses $p_x$ is a proper part of $p_y$ as $PP(p_x, p_y)$.

$PP$ can be defined in terms of $P$.

\[ PP(p_x, p_y) \triangleq P(p_x, p_y) \land \neg P(p_y, p_z) \] (B.4)

**Overlap.** $O$, expresses a relation between parts.

Two parts are said to overlap if they have “something” in common. In classical mereology that ‘something’ is parts. To us parts are spatial entities and these cannot “overlap”.

Instead they can ‘share’ attributes.

\[ O(p_x, p_y) \triangleq \exists a : \mathcal{P} \bullet a \in p_x \land a \in p_y \] (B.5)

**Underlap.** $U$, expresses a relation between parts.

Two parts are said to underlap if there exists a part $p_z$ of which $p_x$ is a part and of which $p_y$ is a part.

\[ U(p_x, p_y) \triangleq \exists p_z : \mathcal{P} \bullet P(p_x, p_z) \land P(p_y, p_z) \] (B.6)

Think of the underlap $p_z$ as an “umbrella” which both $p_x$ and $p_y$ are “under”.

**Over-cross.** $OX$, $p_x$ and $p_y$ are said to over-cross if $p_x$ and $p_y$ overlap and $p_z$ is not part of $p_y$.

\[ OX(p_x, p_y) \triangleq O(p_x, p_y) \land \neg P(p_y, p_z) \] (B.7)

**Under-cross.** $UX$, $p_x$ and $p_y$ are said to under-cross if $p_x$ and $p_y$ underlap and $p_z$ is not part of $p_x$.

---

3 Our notation now is not RSL but a conventional first-order predicate logic notation.
B.3 An Abstract Model of Mereologies

Proper Overlap, \( \text{PO} \), expresses a relation between parts. \( \text{p}_x \) and \( \text{p}_y \) are said to properly overlap if \( \text{p}_x \) and \( \text{p}_y \) over-cross and if \( \text{p}_y \) and \( \text{p}_x \) over-cross.

\[
\text{PO}(\text{p}_x, \text{p}_y) \triangleq \text{OX}(\text{p}_x, \text{p}_y) \land \text{OX}(\text{p}_y, \text{p}_x) \quad \text{(B.9)}
\]

Proper Underlap, \( \text{PU} \), \( \text{p}_x \) and \( \text{p}_y \) are said to properly underlap if \( \text{p}_x \) and \( \text{p}_y \) under-cross and \( \text{p}_y \) and \( \text{p}_x \) under-cross.

\[
\text{PU}(\text{p}_x, \text{p}_y) \triangleq \text{UX}(\text{p}_x, \text{p}_y) \land \text{UX}(\text{p}_y, \text{p}_x) \quad \text{(B.10)}
\]

B.3 An Abstract Model of Mereologies

B.3.1 Parts and Sub-parts

541 We distinguish between atomic and composite parts.

542 Atomic parts do not contain separately distinguishable parts.

543 Composite parts contain at least one separately distinguishable part.

\[
\begin{align*}
\text{type} \quad \\
\text{P} & ::= \text{AP} \mid \text{CP}^4 \\
\text{AP} & ::= \text{mkAP}(\ldots)^5 \\
\text{CP} & ::= \text{mkCP}(\ldots,\text{s-ps:P-set})^6 \quad \text{axiom} \forall \text{mkCP}(\_\text{ps}): \text{CP} \cdot \text{ps} \neq \{\}
\end{align*}
\]

It is the domain analyser who decides what constitutes “the whole”, that is, how parts relate to one another, what constitutes parts, and whether a part is atomic or composite. We refer to the proper parts of a composite part as sub-parts. Figure B.8 illustrates composite and atomic parts. The slanted sans serif uppercase identifiers of Fig. B.8 A1, A2, A3, A4, A5, A6 and C1, C2, C3 are meta-linguistic, that is, they stand for the parts they “decorate”; they are not identifiers of “our system”.

Fig. B.8. Atomic and Composite Parts

---

4 In the RAISE [179] Specification Language, RSL [176], writing type definitions \( X \equiv Y | Z \) means that \( Y \) and \( Z \) are to be disjoint types. In Items 542.–543. the identifiers \( \text{mkAP} \) and \( \text{mkCP} \) are distinct, hence their types are disjoint.

5 In \( Y ::= \text{mkY}(\ldots): y \) values \( (\ldots) \) are marked with the “make constructor” \( \text{mkY} \), cf. [279, 280].

6 In \( Y ::= \text{mkY}(\text{s-w:W,\ldots}) \) \( s_{-w} \) is a “selector function” which when applied to an \( y \), i.e., \( s_{-w}(y) \) identifies the \( W \) element, cf. [279, 280].
B.3.2 No "Infinitely" Embedded Parts

The above syntax, Items 541–543, does not prevent composite parts, \( p \), to contain composite parts, \( p' \), "ad-infinitum"! But we do not wish such "recursively" contained parts!

To express the property that parts are finite we introduce a notion of part derivation.

The part derivation of an atomic part is the empty set.

The part derivation of a composite part, \( p \), \( \text{mkC}(\ldots, ps) \) where \( \ldots \) is left undefined, is the set \( ps \) of sub-parts of \( p \).

\[ \text{pt \_der: P} \rightarrow \text{P-\set} \]
\[ \text{pt \_der(mkAP(\ldots))} \equiv \{\} \]
\[ \text{pt \_der(mkCP(\ldots,ps))} \equiv ps \]

We can also express the part derivation, \( \text{pt \_der(ps)} \) of a set, \( ps \), of parts.

If the set is empty then \( \text{pt \_der(\{\})} \) is the empty set, \{\}.

Let \( \text{mkA(pq)} \) be an element of \( ps \), then \( \text{pt \_der(\{mkA(pq)\} \cup ps')} \) is \( ps' \).

Let \( \text{mkC(pq,ps')} \) be an element of \( ps \), then \( \text{pt \_der(ps' \cup ps)} \) is \( ps' \).

Therefore, to express that a part is finite we postulate a natural number, \( n \), such that a notion of iterated part set derivations lead to an empty set.

An iterated part set derivation takes a set of parts and part set derive that set repeatedly, \( n \) times.

If the result is an empty set, then part \( p \) was finite.

\[ \text{it \_pt \_der: P-\set} \rightarrow \text{Nat} \rightarrow \text{P-\set} \]
\[ \text{it \_pt \_der(ps)(n)} \equiv \begin{cases} ps' & \text{if } n=1 \\ \text{it \_pt \_der(ps')(n-1)} & \text{else} \end{cases} \]

B.3.3 Unique Identifications

Each physical part can be uniquely distinguished for example by an abstraction of its properties at a time of origin. In consequence we also endow conceptual parts with unique identifications.

In order to refer to specific parts we endow all parts, whether atomic or composite, with unique identifications.

We postulate functions which observe these unique identifications, whether as parts in general or as atomic or composite parts in particular.

such that any to parts which are distinct have unique identifications.

\[ \forall p, p': \text{P} \cdot p \neq p' \Rightarrow \text{uid \_UI(p)} \neq \text{uid \_UI(p')} \]
A model for uid_UI can be given. Presupposing subsequent material (on attributes and mereology) — “lumped” into part qualities, pq:PQ, we augment definitions of atomic and composite parts:

\[
\begin{align*}
\text{value} & : \text{uid}_UI(s_{uid:UI},...) \equiv u_i \\
\text{uid}_UI(s_{uid:UI},...) & \equiv u_i
\end{align*}
\]

Figure B.9 illustrates the unique identifications of composite and atomic parts.

No two parts have the same unique identifier.

558 We define an auxiliary function, \( \text{no_parts_uis} \), which applies to any part, \( p \), and yields a pair: the number of sub-parts of the part argument, and the set of unique identifiers of parts within \( p \).

559 \( \text{no_parts_uis} \) is defined in terms of yet an auxiliary function, \( \text{sum_no_parts_uis} \).

\[
\begin{align*}
\text{value} & : \text{no_parts_uis} : P \rightarrow (\text{Nat} \times \text{UI-set}) \rightarrow (\text{Nat} \times \text{UI-set}) \\
\text{no_parts_uis}(\text{mkA}((u_i,...)))((n,u_i)) & \equiv (n+1,u_i\cup\{u_i\}) \\
\text{no_parts_uis}(\text{mkC}((u_i,...),ps))((n,u_i)) & \equiv \\
\text{let} (n',u_i') = \text{sum_no_parts_uis}(ps) \text{ in} \\
(n+n',u_i\cup u_i') & \text{ end} \\
\text{pre: no_infinite_parts}(p) \\
\text{sum_no_parts_uis}(ps)(n,u_i) & \equiv \\
\text{case ps of} \\
\{} & \rightarrow (n,u_i) \\
\{\text{mkA}((u_i,...))\} & \rightarrow \text{sum_no_parts_uis}(ps)(n+1,u_i\cup\{u_i\}) \\
\{\text{mkC}((u_i,...),ps)\} & \rightarrow \text{sum_no_parts_uis}(ps')(1,u_i) \text{ in} \\
\text{let} (n',u_i') = \text{sum_no_parts_uis}(ps')(n+n',u_i\cup u_i') \text{ end} \\
\text{end} \\
\text{pre: } \forall p:P \cdot p \in ps \Rightarrow \text{no_infinite_parts}(p)
\end{align*}
\]

560 That no two parts have the same unique identifier can now be expressed by demanding that the number of parts equals the number of unique identifiers.

\[
\begin{align*}
\text{axiom} & : \forall p:P \cdot \text{let} (n,u_is) = \text{no_parts_uis}(0,\{\}) \text{ in } n = \text{card u_is} \text{ end}
\end{align*}
\]
298  B  MEREOLOGY, A MODEL

B.3.4  Attributes

B.3.4.1  Attribute Names and Values

561  Parts have sets of named attribute values, $attrs:ATTRS$.
562  One can observe attributes from parts.
563  Two distinct parts may share attributes:
   a  For some (one or more) attribute name that is among the attribute names of both parts,
   b  it is always the case that the corresponding attribute values are identical.

\[
\begin{align*}
561. \text{ANm, AVAL, ATTRS = ANm \rightarrow AVAL} \\
564. \text{SA, CA = AVAL} \\
565. \text{EA = ANm} \\
566. \text{ME = UI-set}
\end{align*}
\]

The function $\text{trans}_{\text{adj}}$ is defined in Sect. B.4.4 on Page 300.

B.3.4.2  Attribute Categories

We define some auxiliary functions:

564  $\mathcal{A}_{\text{df}}$ applies to $attrs:ATTRS$ and yields a grouping $(sa_1, sa_2, \ldots, sa_n)^7$ of static attribute values.
565  $\mathcal{C}_{\text{df}}$ applies to $attrs:ATTRS$ and yields a grouping $(ca_1, ca_2, \ldots, ca_n)^8$ of controllable attribute values.
566  $\mathcal{E}_{\text{df}}$ applies to $attrs:ATTRS$ and yields a set, $\{eA_1,eA_2,\ldots,eA_{n_e}\}^9$ of external attribute names.

\[
\begin{align*}
564. \mathcal{A}_{\text{df}}: ATTRS &\rightarrow SA \\
565. \mathcal{C}_{\text{df}}: ATTRS &\rightarrow CA \\
566. \mathcal{E}_{\text{df}}: ATTRS &\rightarrow EA
\end{align*}
\]

The attribute names of static, controllable and external attributes do not overlap and together make up the attribute names of $attrs$.

B.3.5  Mereology

In order to illustrate other than the within and adjacency part relations we introduce the notion of mereology. Figure B.10 on the next page illustrates a mereology between parts. A specific mereology-relation is, visually, a $\bullet\rightarrow\bullet$ line that connects two distinct parts.

567  The mereology of a part is a set of unique identifiers of other parts.

\[
\begin{align*}
567. \text{ME = UI-set}
\end{align*}
\]

We may refer to the connectors by the two element sets of the unique identifiers of the parts they connect. For example with respect to Fig. B.10 on the facing page:

---

7  where $\{sa_1, sa_2, \ldots, sa_n\} \subseteq \text{rng \ attr}$

8  where $\{ca_1, ca_2, \ldots, ca_n\} \subseteq \text{rng \ attr}$

9  where $\{eA_1, eA_2, \ldots, eA_{n_e}\} \subseteq \text{dom \ attr}$
B.4 Some Part Relations

Fig. B.10. Mereology: Relations between Parts

- \{ci_1, ci_3\},
- \{ai_2, ai_3\},
- \{ai_6, ci_1\},
- \{ai_3, ci_1\},
- \{ai_6, ai_5\} and
- \{ai_1, ci_1\}.

B.3.6 The Model

A part value has a part sort name and is either the value of an atomic part or of an abstract composite part.

An atomic part value has a part quality value.

An abstract composite part value has a part quality value and a set of at least one or more part values.

A part quality value consists of a unique identifier, a mereology, and a set of one or more attribute named attribute values.

We now assume that parts are not “recursively infinite”, and that all parts have unique identifiers.

B.4 Some Part Relations

B.4.1 ‘Immediately Within’

One part, p, is said to be immediately within, imm_within(p, p’), another part, if p’ is a composite part and p is observable in p’.

value

573. imm_within: P \times P \rightarrow \text{Bool}
573. imm_within(p, p’) \equiv
573. \text{case } p’ \text{ of}
573. \quad \text{__mkA(__ps)) } \rightarrow p \in ps,
573. \quad \text{__mkC(__ps)) } \rightarrow p \in ps,
573. \quad \_ \rightarrow \text{false}
573. \text{end}
B.4.2 ‘Transitive Within’

We can generalise the ‘immediate within’ property.

A part, p, is transitively within a part p′, trans_within(p,p′),

a either if p, is immediately within p′

b or

c if there exists a (proper) composite part p″ of p′ such that trans_within(p″,p).

B.4.3 ‘Adjacency’

Two parts, p,p′, are said to be immediately adjacent, imm_adj(p,p′)(c), to one another, in a composite part c, such that p and p′ are distinct and observable in c.

B.4.4 Transitive ‘Adjacency’

We can generalise the immediate ‘adjacent’ property.

Two parts, p′,p″, of a composite part, p, are trans_adj(p′, p″) in p

a either if imm_adj(p′,p″)(p),

b or if there are two p‴ and p‴‴ such that

i p‴ and p‴‴ are immediately adjacent parts of p and

ii p is equal to p‴ or p‴‴ is properly within p and p′ is equal to p‴‴ or p‴‴ is properly within p′

We leave the formalisation to the reader.

B.5 Satisfaction

We shall sketch a proof that the model of Sect. B.3, satisfies, i.e., is a model of, the axioms of Sect. B.2.
B.5 Satisfaction

B.5.1 A Proof Sketch

We assign
- $P$ as the meaning of $\mathcal{P}$,
- $\text{ATR}$ as the meaning of $\mathcal{A}$,
- $\text{imm}_{\text{within}}$ as the meaning of $\mathcal{I}_{\mathcal{P}}$,
- $\text{trans}_{\text{within}}$ as the meaning of $\mathcal{I}_{\mathcal{P}}$,
- $\epsilon : \text{ATTR} \times \text{ATTR-set} \rightarrow \text{Bool}$ as the meaning of $\epsilon : \mathcal{A} \times \mathcal{P} \rightarrow \text{Bool}$ and
- $\text{sharing}$ as the meaning of $\mathcal{O}$.

With the above assignments it is now easy to prove that the other axiom-operators $\forall$, $\exists\mathcal{O}$, $\forall\mathcal{U}$, $\mathcal{O}X$ and $\forall\mathcal{UX}$ can be modeled by means of $\text{imm}_{\text{within}}$, $\text{within}$, $\text{ATTR} \times \text{ATTR-set} \rightarrow \text{Bool}$ and $\text{sharing}$. 
In this appendix we recall the four language tools of the domain analysis & description: (i) the calculi of analysis and description prompts; (ii) the ‘language’ of explaining domain analysis & description; (iii) the RSL: Raise Specification Language, and (iv) the ‘language’ of domains.

Usually mathematics, in many of its shades and forms are deployed in describing properties of nature, as when pursuing physics, Usually the formal specification languages of computer & computing science have a precise semantics and a consistent proof system. To have these properties those languages must deal with computable objects. Domains are not computable.

C.1 The Domain Analysis & Description Calculi

We separate the calculi into two: the analysis functions, and the description functions. None of these are computable functions as they have no formal basis. They are tools in helping us to achieve a formal, computable basis on which to understand the analysed & described domains.

C.1.1 The Analysis Calculus

Use of the analysis language is not written down. It consists of a number of single, usually is, or has, prefixed domain analysis prompt and domain description prompt names. The domain analysis prompts are:

<table>
<thead>
<tr>
<th>Domain Analysis Prompts</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_animal, 54</td>
</tr>
<tr>
<td>is_artefactual_composite, 56</td>
</tr>
<tr>
<td>is_artefactual_atomic, 56</td>
</tr>
<tr>
<td>is_artefact, 51</td>
</tr>
<tr>
<td>is_atomic, 56</td>
</tr>
<tr>
<td>is_composite, 56</td>
</tr>
<tr>
<td>is_compound_structure, 52</td>
</tr>
<tr>
<td>is_conjoin, 57</td>
</tr>
<tr>
<td>is_continuous, 49</td>
</tr>
<tr>
<td>is_discrete, 48</td>
</tr>
<tr>
<td>is_endurant, 47</td>
</tr>
<tr>
<td>is_entity, 43</td>
</tr>
<tr>
<td>is_human, 54</td>
</tr>
<tr>
<td>is_living_species, 51</td>
</tr>
<tr>
<td>is_material_parts_conjoin, 59</td>
</tr>
<tr>
<td>is_material, 49</td>
</tr>
<tr>
<td>is_natural_atomic, 56</td>
</tr>
<tr>
<td>is_natural_composite, 56</td>
</tr>
<tr>
<td>is_natural_part, 51</td>
</tr>
<tr>
<td>is_part_materials_conjoin, 57</td>
</tr>
<tr>
<td>is_part_parts_conjoin, 59</td>
</tr>
<tr>
<td>is_perdurant, 48</td>
</tr>
<tr>
<td>is_physical_part, 50</td>
</tr>
<tr>
<td>is_plant, 53</td>
</tr>
<tr>
<td>is_set_structure, 53</td>
</tr>
<tr>
<td>is_structure, 50</td>
</tr>
<tr>
<td>has_monitorable_attributes, 126</td>
</tr>
<tr>
<td>is_physical_attribute, 205, 107</td>
</tr>
</tbody>
</table>
They apply to phenomena in the domain, that is, to “the world out there”! Except for the `analyse···` and `attribute types` functions, these queries result in truth values; the `analyse···` result in the domain scientist cum engineer noting down, in memory or in typed form, suggestive names [of endurant sorts]; and `attribute types` results in suggestive names [of attribute types]. The truth-valued queries directs, as we shall see, the domain scientist cum engineer to either further analysis or to “issue” some domain description prompts.

C.1.2 The Description Calculus

The ‘name’-valued queries help the human analyser to formulate the result of domain description prompts:

Again they apply to phenomena in the domain, that is, to “the world out there”! In this case they result in RSL-Text!

The description language is RSL+. It is a basically applicative subset of RSL [176], that is: no assignable variables. Also we omit RSL’s elaborate scheme, class, object notions.

- **Endurants:**
  - `obs_E`, dfn. 2, [o] pg. 66, dfn. 2, [s] pg. 66
- **Unique Identifiers:**
  - `uid_P`, dfn. 9, [u] pg. 86
- **Mereologies:**
  - `mero_P`, dfn. 10, [m] pg. 90
- **Attributes:**
  - `attr_A`, dfn. 11, [a] pg. 95
We refer, generally, to all these functions as observer functions. They are defined by the analyser cum describer when “applying” description prompts. That is, they should be considered user-defined. In our examples we use the non-bold-faced observer function names.

C.2 The Language of Explaining Domain Analysis & Description

**English, Philosophy and Discrete Mathematics Notation**

In explaining the *analysis & description prompts* we use a natural language which contains terms and phrases typical of (i) the technical language of *computer & computing science*, and (ii) the language of *philosophy*, more specifically *ontology*, and discrete mathematics notation. The reason for the former should be obvious. The reason for the latter is given as follows: We are, on one hand, dealing with real, actual segments of domains characterised by their basis in nature, in economics, in technologies, etc., that is, in informal “worlds”, and, on the other hand, we aim at a formal understanding of those “worlds”. There is, in other words, the task of explaining how we observe those “worlds”, and that is what brings us close to some issues well-discussed in *philosophy*.

C.3 The RSL: Raise Specification Language

RSL is the target language into which the domain description prompts express their results. The author has been involved in both the development, research into and extensive use of both VDM and RAISE/RSL. He instigated the mainly UK/Danish project that led to RAISE/RSL. From around 1993 he has used RSL on an almost daily basis.

The RAISE Specification Language is basically a model-oriented specification language. Bases for RSL are VDM [88, 89, 154], discrete mathematics, and CSP [238]. For initial specifications, like, e.g., domain descriptions, we advice to focus on the functional, i.e., the applicative aspects of RSL.

The prime references to the RAISE Method and the RSL, Raise Specification Language, are [179, 176]. Short introductions to RAISE and RSL are [177, 168, 172, George et al.].

Early publications: [118, 131, 240, 132, 275, 133, 145, 183, 381, 146, Dandanell et al.]; theoretical investigations [290, 291, Milner]; case studies [180, 2001]; and by the current author [39, 40, 41, Bjørner].

Chris W. George, is one of the masterminds, since the mid-to-late 1980s, of RAISE, focusing very much on correctness issues, is the prime author of most of these papers: [162, 296, 298, 163, 164, 216, 165, 169, 378, 265, 175, 166, 171, 167, 2, 135, 84, 3, 311, 168, 174, 168, 357, 173, 356, 119, 120, 120, 312, 140, Chris W. George et al.].

Klaus Havelund, who was with the RAISE project at the Danish industrial partner, CR, in its early days, besides co-authoring [176, 179], was a prime author of many of the RAISE project technical reports – as well of these early publications: [297, 170, 86, 201].


There are some Web pages: RAISE Tools: https://raisetools.github.io/ and a RAISE Repository: https://github.com/raisetools. From here one should be able to download the RAISE Tools.

C.4 The Language of Domains

We consider a domain through the *semiotic looking glass* of its *syntax* and its *semantics*; we shall not consider here its possible *pragmatics*. By “*its syntax*” we shall mean the form and “*contents*”, i.e., the
external and internal qualities of the endurants of the domain, i.e., those entities that endure. By “its semantics” we shall, by a transcendental deduction, mean the perdurants: the actions, the events, and the behaviours that center on the the endurants and that otherwise characterise the domain.
This is an ultra-short introduction to the RAISE Specification Language, RSL.

D.1 Types

The reader is kindly asked to study first the decomposition of this section into its sub-parts and sub-sub-parts.

D.1.1 Type Expressions

Type expressions are expressions whose value are type, that is, possibly infinite sets of values (of “that” type).

D.1.1.1 Atomic Types

Atomic types have (atomic) values. That is, values which we consider to have no proper constituent (sub-)values, i.e., cannot, to us, be meaningfully “taken apart”.

RSL has a number of built-in atomic types. There are the Booleans, integers, natural numbers, reals, characters, and texts.

D.1.1.0.1 Basic Types:

<table>
<thead>
<tr>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

D.1.1.2 Composite Types

Composite types have composite values. That is, values which we consider to have proper constituent (sub-)values, i.e., can, to us, be meaningfully “taken apart”.

From these one can form type expressions: finite sets, infinite sets, Cartesian products, lists, maps, etc.

Let A, B and C be any type names or type expressions, then:
D.1.1.2.0.1 Composite Type Expressions:

- $A\text{-set}$
- $A\text{-infset}$
- $A \times B \times \ldots \times C$
- $A^*$
- $A^\omega$
- $A \mathbin{\|} B$
- $A \leadsto B$
- $A \mid B \mid \ldots \mid C$
- $\text{mk}\_\text{id}(\text{sel}_a:A,\ldots,\text{sel}_b:B)$
- $\text{sel}_a:A \ldots \text{sel}_b:B$

The following are generic type expressions: false and true.
1. The integer type on integers, $-2, -1, 0, 1, 2, \ldots$
2. The natural number type of positive integer values $0, 1, 2, \ldots$
3. The real number type of real values, i.e., values whose numerals can be written as an integer, followed by a period ("."), followed by a natural number (the fraction).
4. The character type of character values "a", "bb", ...
5. The text type of character string values "aa", "aaa", ..., "abc", ...
6. The set type of finite cardinality set values.
7. The set type of infinite and finite cardinality set values.
8. The Cartesian type of Cartesian values.
9. The list type of finite length list values.
10. The list type of infinite and finite length list values.
11. The map type of finite definition set map values.
12. The function type of total function values.
13. The function type of partial function values.
14. The postulated disjoint union of types $A, B, \ldots, C$.
15. The record type of $\text{mk}\_\text{id}$-named record values $\text{mk}\_\text{id}(av,\ldots,bv)$, where $av, \ldots, bv$, are values of respective types. The distinct identifiers $\text{sel}_a, \ldots, \text{sel}_b$, designate selector functions.
16. The record type of unnamed record values $(av,\ldots,bv)$, where $av, \ldots, bv$, are values of respective types. The distinct identifiers $\text{sel}_a, \ldots, \text{sel}_b$, designate selector functions.

D.1.2 Type Definitions

D.1.2.1 Concrete Types

Types can be concrete in which case the structure of the type is specified by type expressions:

D.1.2.1.0.1 Type Definition:

\begin{verbatim}
   type
   A = TypeExpr
\end{verbatim}

Some schematic type definitions are:
D.1.2.1.0.2 Variety of Type Definitions:

1. Type\_name = Type\_expr without | or subtypes */
2. Type\_name = Type\_expr । Type\_expr । ... । Type\_expr
3. Type\_name ==
   mk\_id\_1(s\_a1:Type\_name\_a1,...,s\_ai:Type\_name\_ai) ।...
   mk\_id\_n(s\_z1:Type\_name\_z1,...,s\_zk:Type\_name\_zk)
4. Type\_name :: sel\_a:Type\_name\_a । ... । sel\_z:Type\_name\_z
5. Type\_name = {v:Type\_name\_\': P(v)}

where a form of [2–3] is provided by combining the types:

D.1.2.1.0.3 Record Types:

A = mk\_id\_1(s\_a1:A\_1,...,s\_ai:A\_i)
B = mk\_id\_2(s\_b1:B\_1,...,s\_bj:B\_j)
... 
Z = mk\_id\_n(s\_z1:Z\_1,...,s\_zk:Z\_k)

Types A, B, ..., Z are disjoint, i.e., shares no values, provided all mk\_id\_k are distinct and due to the use of the disjoint record type constructor ==.

axiom
∀ a1:A\_1, a2:A\_2, ..., ai:AI ·
   s\_a1(mk\_id\_1(a1,a2,...,ai))=a1 ∧ s\_a2(mk\_id\_1(a1,a2,...,ai))=a2 ∧
   ... ∧ s\_ai(mk\_id\_1(a1,a2,...,ai))=ai ∧
∀ a:A · let mk\_id\_1(a1,a2',...,ai') = a in
   a' = s\_a1(a) ∧ a2' = s\_a2(a) ∧ ... ∧ ai' = s\_ai(a) end

Note: Values of type A, where that type is defined by A::B×C×D, can be expressed A(b,c,d) for b:B, c:D, d:D.

D.1.2.2 Subtypes

In RSL, each type represents a set of values. Such a set can be delimited by means of predicates. The set of values b which have type B and which satisfy the predicate \( P \), constitute the subtype A:

D.1.2.2.0.1 Subtypes:

\[ A = \{ b: B \land P(b) \} \]

D.1.2.3 Sorts — Abstract Types

Types can be (abstract) sorts in which case their structure is not specified:

D.1.2.3.0.1 Sorts:

\[ A, B, ... , C \]

D.2 The RSL Predicate Calculus

D.2.1 Propositional Expressions

Let identifiers (or propositional expressions) a, b, ..., c designate Boolean values (true or false [or chaos]). Then:
Propositional Expressions:

false, true
a, b, ..., c \sim a, a \land b, a \lor b, a \Rightarrow b, a = b, a \neq b

are propositional expressions having Boolean values. \sim, \land, \lor, \Rightarrow, =, \neq and \Box are Boolean connectives (i.e., operators). They can be read as: not, and, or, if then (or implies), equal and not equal.

Simple Predicate Expressions

Let identifiers (or propositional expressions) a, b, ..., c designate Boolean values, let x, y, ..., z (or term expressions) designate non-Boolean values and let i, j, ..., k designate number values, then:

Simple Predicate Expressions:

\forall x:X \cdot P(x) \\
\exists y:Y \cdot Q(y) \\
\exists ! z:Z \cdot R(z)

are quantified expressions — also being predicate expressions. They are “read” as: For all x (values in type X) the predicate P(x) holds; there exists (at least) one y (value in type Y) such that the predicate Q(y) holds; and there exists a unique z (value in type Z) such that the predicate R(z) holds.

Concrete RSL Types: Values and Operations

Arithmetic

Arithmetic:

\begin{align*}
\text{type} \quad & \text{Nat, Int, Real} \\
\text{value} \quad & +, -, \ast: \text{Nat} \times \text{Nat} \rightarrow \text{Nat} | \text{Int} \times \text{Int} \rightarrow \text{Int} | \text{Real} \times \text{Real} \rightarrow \text{Real} \\
\div & : \text{Nat} \times \text{Nat} \rightarrow \text{Nat} | \text{Int} \times \text{Int} \rightarrow \text{Int} | \text{Real} \times \text{Real} \rightarrow \text{Real} \\
\lt, \leq, =, \neq, \geq, \gt & : (\text{Nat} | \text{Int} | \text{Real}) \rightarrow (\text{Nat} | \text{Int} | \text{Real})
\end{align*}

Set Expressions

Set Enumerations

Let the below a’s denote values of type A, then the below designate simple set enumerations:

Set Enumerations:

\begin{align*}
\{\}, \{a\}, \{e_1, e_2, ..., e_n\}, ... \in & \text{A-set} \\
\{\}, \{a\}, \{e_1, e_2, ..., e_n\}, ..., \{e_1, e_2, ...\} \in & \text{A-infset}
\end{align*}

Set Comprehension

The expression, last line below, to the right of the \equiv, expresses set comprehension. The expression “builds” the set of values satisfying the given predicate. It is abstract in the sense that it does not do so by following a concrete algorithm.
D.3.2.2.0.1 **Set Comprehension:**

**type**

\[ A, B \]

\[ P = A \rightarrow \text{Bool} \]

\[ Q = A \leadsto B \]

**value**

comprehend: \( A^{\text{infset}} \times P \times Q \rightarrow B^{\text{infset}} \)

\[ \text{comprehend}(s,P,Q) \equiv \{ Q(a) | a: A \rightarrow a \in s \land P(a) \} \]

D.3.3 **Cartesian Expressions**

D.3.3.1 **Cartesian Enumerations**

Let \( e \) range over values of Cartesian types involving \( A, B, \ldots, C \), then the below expressions are simple Cartesian enumerations:

D.3.3.1.0.1 **Cartesian Enumerations:**

**type**

\[ A, B, \ldots, C \]

\[ A \times B \times \ldots \times C \]

**value**

\( (e_1,e_2,\ldots,e_n) \)

D.3.4 **List Expressions**

D.3.4.1 **List Enumerations**

Let \( a \) range over values of type \( A \), then the below expressions are simple list enumerations:

D.3.4.1.0.1 **List Enumerations:**

\[ \{ \langle \rangle, \langle e \rangle, \ldots, \langle e_1,e_2,\ldots,en \rangle, \ldots \} \in A^* \]

\[ \{ \langle \rangle, \langle e \rangle, \ldots, \langle e_1,e_2,\ldots,en \rangle, \ldots, \langle e_1,e_2,\ldots,en,\ldots \rangle, \ldots \} \in A^{\omega} \]

\[ \langle a_1 \ldots a_n \rangle \]

The last line above assumes \( a_i \) and \( a_j \) to be integer-valued expressions. It then expresses the set of integers from the value of \( e_i \) to and including the value of \( e_j \). If the latter is smaller than the former, then the list is empty.

D.3.4.2 **List Comprehension**

The last line below expresses list comprehension.

D.3.4.2.0.1 **List Comprehension:**

**type**

\[ A, B, P = A \rightarrow \text{Bool}, Q = A \leadsto B \]

**value**

comprehend: \( A^{\omega} \times P \times Q \leadsto B^{\omega} \)

\[ \text{comprehend}(l,P,Q) \equiv \{ Q(l(i)) | i \text{ in } \langle 1..\text{len l} \rangle \cdot P(l(i)) \} \]
D.3.5 Map Expressions

D.3.5.1 Map Enumerations

Let (possibly indexed) \( u \) and \( v \) range over values of type \( T_1 \) and \( T_2 \), respectively, then the below expressions are simple map enumerations:

D.3.5.1.0.1 Map Enumerations:

\[
\begin{align*}
\text{type} & \quad T_1, T_2 \\
M & \quad = T_1 \rightarrow T_2 \\
\text{value} & \quad u, u_1, u_2, ..., u_n : T_1, v, v_1, v_2, ..., v_n : T_2 \\
& \quad [], [u \mapsto v], ..., [u_1 \mapsto v_1, u_2 \mapsto v_2, ..., u_n \mapsto v_n] \quad \forall \in M
\end{align*}
\]

D.3.5.2 Map Comprehension

The last line below expresses map comprehension:

D.3.5.2.0.1 Map Comprehension:

\[
\begin{align*}
\text{type} & \quad U, V, X, Y \\
M & \quad = U \rightarrow V \\
F & \quad = U \rightarrow X \\
G & \quad = V \rightarrow Y \\
P & \quad = U \rightarrow \text{Bool} \\
\text{value} & \quad \text{comprehend}: M \times F \times G \times P \rightarrow (X \rightarrow Y) \\
\text{comprehend}(m, F, G, P) & \equiv \\
& \quad [F(u) \mapsto G(m(u)) \mid u : U \cdot u \in \text{dom } m \land P(u)]
\end{align*}
\]

D.3.6 Set Operations

D.3.6.1 Set Operator Signatures

D.3.6.1.0.1 Set Operations:

\[
\begin{align*}
\text{value} & \quad \in: A \times A\text{-inset} \rightarrow \text{Bool} \\
\notin: A \times A\text{-inset} \rightarrow \text{Bool} \\
\cup: A\text{-inset} \times A\text{-inset} \rightarrow A\text{-inset} \\
\cup: (A\text{-inset})\text{-inset} \rightarrow A\text{-inset} \\
\cap: A\text{-inset} \times A\text{-inset} \rightarrow A\text{-inset} \\
\cap: (A\text{-inset})\text{-inset} \rightarrow A\text{-inset} \\
\setminus: A\text{-inset} \times A\text{-inset} \rightarrow A\text{-inset} \\
\subseteq: A\text{-inset} \times A\text{-inset} \rightarrow \text{Bool} \\
\subseteq: A\text{-inset} \times A\text{-inset} \rightarrow \text{Bool} \\
\neq: A\text{-inset} \times A\text{-inset} \rightarrow \text{Bool} \\
\neq: A\text{-inset} \times A\text{-inset} \rightarrow \text{Bool} \\
\text{card}: A\text{-inset} \rightarrow \text{Nat}
\end{align*}
\]
D.3.6.2 Set Examples

D.3.6.2.0.1 Set Examples:

Examples
- \( a \in \{a, b, c\} \)
- \( a \notin \emptyset \)
- \( \{a, b, c\} \cup \{a, b, d, e\} = \{a, b, c, d, e\} \)
- \( \cup\{\{a\}, \{a, b\}, \{a, d\}\} = \{a\} \)
- \( \{a, b, c\} \cap \{c, d\} = \{a\} \)
- \( \{a, b, c\} \setminus \{c, d\} = \{a\} \)
- \( \{a, b\} \subseteq \{a, b, c\} \)
- \( \{a, b, c\} = \{a, b, c\} \)
- \( \{a, b, c\} \neq \{a, b\} \)
- \( \text{card} \{\} = 0, \text{card} \{a, b, c\} = 3 \)

D.3.6.3 Informal Explication

- \( \in : \) The membership operator expresses that an element is a member of a set.
- \( \notin : \) The nonmembership operator expresses that an element is not a member of a set.
- \( \cup : \) The infix union operator. When applied to two sets, the operator gives the set whose members are in either or both of the two operand sets.
- \( \cup : \) The distributed prefix union operator. When applied to a set of sets, the operator gives the set whose members are in some of the operand sets.
- \( \cap : \) The infix intersection operator. When applied to two sets, the operator gives the set whose members are in both of the two operand sets.
- \( \cap : \) The prefix distributed intersection operator. When applied to a set of sets, the operator gives the set whose members are in some of the operand sets.
- \( \setminus : \) The set complement (or set subtraction) operator. When applied to two sets, the operator gives the set whose members are those of the left operand set which are not in the right operand set.
- \( \subseteq : \) The proper subset operator expresses that all members of the left operand set are also in the right operand set.
- \( \subset : \) The proper subset operator expresses that all members of the left operand set are also in the right operand set, and that the two sets are not identical.
- \( = : \) The equal operator expresses that the two operand sets are identical.
- \( \neq : \) The nonequal operator expresses that the two operand sets are not identical.
- \( \text{card} : \) The cardinality operator gives the number of elements in a finite set.

D.3.6.4 Set Operator Definitions

The operations can be defined as follows (\( \equiv \) is the definition symbol):

D.3.6.4.0.1 Set Operation Definitions:

**value**

\[
\begin{align*}
\text{s}' \cup \text{s}'' & \equiv \{ a \mid \text{a:A} \land \text{a} \in \text{s}' \lor \text{a} \in \text{s}'' \} \\
\text{s}' \cap \text{s}'' & \equiv \{ a \mid \text{a:A} \land \text{a} \in \text{s}' \land \text{a} \in \text{s}'' \} \\
\text{s}' \setminus \text{s}'' & \equiv \{ a \mid \text{a:A} \land \text{a} \in \text{s}' \land \text{a} \notin \text{s}'' \} \\
\text{s}' & \subseteq \text{s}'' \equiv \forall \text{a:A} \cdot \text{a} \in \text{s}' \Rightarrow \text{a} \in \text{s}'' \\
\text{s}' & \subset \text{s}'' \equiv \text{s}' \subseteq \text{s}'' \land \exists \text{a:A} \cdot \text{a} \in \text{s}'' \land \text{a} \notin \text{s}' \\
\text{s}' = \text{s}'' & \equiv \forall \text{a:A} \cdot \text{a} \in \text{s}' \equiv \text{a} \in \text{s}'' \equiv \exists \text{s} \subseteq \text{s}' \wedge \text{s} \subseteq \text{s} \\
\text{s}' \neq \text{s}'' & \equiv \text{s}' \cap \text{s}'' \neq \{} 
\end{align*}
\]
AN RSL PRIMER

D.3.7 Cartesian Operations

D.3.7.0.0.1 Cartesian Operations:

**type**

\[ A, B, C \]

\[ g0: G0 = A \times B \times C \]

\[ g1: G1 = (A \times B \times C) \]

\[ g2: G2 = (A \times B) \times C \]

\[ g3: G3 = A \times (B \times C) \]

**value**

\[ va:A, vb:B, vc:C, vd:D \]

\[ (va, vb, vc):G0 \]

\[ (va, vb, vc):G1 \]

\[ ((va, vb), vc):G2 \]

\[ (va3, (vb3, vc3)):G3 \]

**decomposition expressions**

\[ let (a1, b1, c1) = g0, (a1', b1', c1') = g1 \] in ...

\[ let ((a2, b2), c2) = g2 \] in ...

\[ let (a3, (b3, c3)) = g3 \] in ...

D.3.8 List Operations

D.3.8.1 List Operator Signatures

D.3.8.1.0.1 List Operations:

**value**

\[ \text{hd: } A^\omega \rightsquigarrow A \]

\[ \text{tl: } A^\omega \rightsquigarrow A^\omega \]

\[ \text{len: } A^\omega \rightsquigarrow \text{Nat} \]

\[ \text{inds: } A^\omega \rightarrow \text{Nat-inset} \]

\[ \text{elems: } A^\omega \rightarrow A\text{-inset} \]

\[ .(\cdot): A^\omega \times \text{Nat} \rightarrow A \]

\[ ^\sim: A^* \times A^\omega \rightarrow A^\omega \]

\[ =:\ A^\omega \times A^\omega \rightarrow \text{Bool} \]

\[ \neq:\ A^\omega \times A^\omega \rightarrow \text{Bool} \]

D.3.8.2 List Operation Examples

D.3.8.2.0.1 List Examples:

**examples**

\[ \text{hd}[a1,a2,\ldots,am] = a1 \]

\[ \text{tl}[a1,a2,\ldots,am] = (a2,\ldots,am) \]

\[ \text{len}(a1,a2,\ldots,am) = m \]

\[ \text{inds}(a1,a2,\ldots,am) = \{1,2,\ldots,m\} \]

\[ \text{elems}(a1,a2,\ldots,am) = \{a1,a2,\ldots,am\} \]

\[ \langle a1,a2,\ldots,am \rangle(i) = ai \]

\[ \langle a,b,c \rangle \tilde{\sim} \langle a,b,d \rangle = \langle a,b,c,a,b,d \rangle \]

\[ \langle a,b,c \rangle = \langle a,b,c \rangle \]

\[ \langle a,b,c \rangle \neq \langle a,b,d \rangle \]
D.3.8.3 Informal Explication

- **hd**: Head gives the first element in a nonempty list.
- **tl**: Tail gives the remaining list of a nonempty list when Head is removed.
- **len**: Length gives the number of elements in a finite list.
- **inds**: Indices give the set of indices from 1 to the length of a nonempty list. For empty lists, this set is the empty set as well.
- **elems**: Elements gives the possibly infinite set of all distinct elements in a list.
- **ℓ**: Indexing with a natural number, i larger than 0, into a list ℓ having a number of elements larger than or equal to i, gives the ith element of the list.
- **ˆ**: Concatenates two operand lists into one. The elements of the left operand list are followed by the elements of the right. The order with respect to each list is maintained.
- **=:** The equal operator expresses that the two operand lists are identical.
- **/=**: The nonequal operator expresses that the two operand lists are not identical.

The operations can also be defined as follows:

D.3.8.4 List Operator Definitions

D.3.8.4.0.1 List Operator Definitions:

```
value

is_finite_list: A^ω → Bool

len q ≡
  case is_finite_list(q) of
  true → if q = ⟨⟩ then 0 else 1 + len tl q end,
  false → chaos end

inds q ≡
  case is_finite_list(q) of
  true → { i | i:Nat • 1 ≤ i ≤ len q },
  false → { i | i:Nat • i/=0 } end

elems q ≡ { q(i) | i:Nat • i ∈ inds q }

q(i) ≡
  if i=1
  then
    if q/=⟨⟩
    then let a:A,q':'Q • q=(a)ˆq' in a end
    else chaos end
  else q(i-1) end

fq ~ iq ≡
  ⟨ if 1 ≤ i ≤ len fq then fq(i) else iq(i- len fq) end
  | i:Nat • if len iq /= chaos then i ≤ len fq+len end ⟩
pre is_finite_list(fq)

iq' = iq'' ≡
  inds iq' = inds iq'' ∧ ∀ i:Nat • i ∈ inds iq' ⇒ iq'(i) = iq''(i)

iq' /= iq'' ≡ ~ (iq' = iq'')
```
D.3.9 Map Operations

D.3.9.1 Map Operator Signatures and Map Operation Examples

value

\[ m(a) : M \rightarrow A \sim B, \; m(a) = b \]

\textbf{dom:} M \rightarrow \text{A-infset} \quad [\text{domain of map}]

\[ \text{dom} \; \{a_1 \mapsto b_1, a_2 \mapsto b_2, \ldots, a_n \mapsto b_n\} = \{a_1, a_2, \ldots, a_n\} \]

\textbf{rng:} M \rightarrow \text{B-infset} \quad [\text{range of map}]

\[ \text{rng} \; \{a_1 \mapsto b_1, a_2 \mapsto b_2, \ldots, a_n \mapsto b_n\} = \{b_1, b_2, \ldots, b_n\} \]

\[ \vdash : M \times M \rightarrow M \quad [\text{override extension}] \]

\[ [a \mapsto b, a' \mapsto b', a'' \mapsto b''] \vdash [a' \mapsto b', a'' \mapsto b'] = [a \mapsto b, a' \mapsto b'', a'' \mapsto b'] \]

\[ \cup : M \times M \rightarrow M \quad [\text{merge}] \]

\[ [a \mapsto b, a' \mapsto b', a'' \mapsto b''] \cup [a''' \mapsto b'''] = [a \mapsto b, a' \mapsto b', a'' \mapsto b'', a''' \mapsto b'''] \]

\[ \setminus : M \times A \rightarrow M \quad [\text{restriction by}] \]

\[ [a \mapsto b, a' \mapsto b', a'' \mapsto b''] \setminus \{a\} = [a' \mapsto b', a'' \mapsto b'''] \]

\[ / : M \times A \rightarrow M \quad [\text{restriction to}] \]

\[ [a \mapsto b, a' \mapsto b', a'' \mapsto b''] / \{a', a''\} = [a' \mapsto b', a'' \mapsto b'''] \]

\[ =, \neq : M \times M \rightarrow \text{Bool} \]

\[ \circ : (A \mapsto B) \times (B \mapsto C) \rightarrow (A \mapsto C) \quad [\text{composition}] \]

\[ [a \mapsto b, a' \mapsto b'] \circ [b \mapsto c, b' \mapsto c'] = [a \mapsto c, a' \mapsto c'] \]

D.3.9.2 Map Operation Explication

- \textit{m(a):} Application gives the element that \( a \) maps to in the map \( m \).
- \textit{dom:} Domain/Definition Set gives the set of values which maps to in a map.
- \textit{rng:} Range/Image Set gives the set of values which are mapped to in a map.
- \textit{\vdash:} Override/Extend. When applied to two operand maps, it gives the map which is like an override of the left operand map by all or some “pairings” of the right operand map.
- \textit{\cup:} Merge. When applied to two operand maps, it gives a merge of these maps.
- \textit{\setminus:} Restriction. When applied to two operand maps, it gives the map which is a restriction of the left operand map to the elements that are not in the right operand set.
- \textit{/:} Restriction. When applied to two operand maps, it gives the map which is a restriction of the left operand map to the elements of the right operand set.
- \textit{=, \neq:} The equal operator expresses that the two operand maps are identical.
- \textit{\circ:} Composition. When applied to two operand maps, it gives the map from definition set elements of the left operand map, \( m_1 \), to the range elements of the right operand map, \( m_2 \), such that if \( a \) is in the definition set of \( m_1 \) and maps into \( b \), and if \( b \) is in the definition set of \( m_2 \) and maps into \( c \), then \( a \) in the composition, maps into \( c \).

D.3.9.3 Map Operation Redefinitions

The map operations can also be defined as follows:
D.3.9.3.0.1 Map Operation Redefinitions:

value

\( \text{rng } m \equiv \{ m(a) | a : A \cdot a \in \text{dom } m \} \)

\( m_1 \upharpoonright m_2 \equiv \left[ a \mapsto b | a : A, b : B \cdot a \in \text{dom } m_1 \land b = m_1(a) \lor a \in \text{dom } m_2 \land b = m_2(a) \right] \)

\( m_1 \cup m_2 \equiv \left[ a \mapsto b | a : A, b : B \cdot a \in \text{dom } m_1 \land b = m_1(a) \lor a \in \text{dom } m_2 \land b = m_2(a) \right] \)

\( m \setminus s \equiv \left[ a \mapsto m(a) | a : A, a \in \text{dom } m \setminus s \right] \)

\( m / s \equiv \left[ a \mapsto m(a) | a : A, a \in \text{dom } m \cap s \right] \)

\( m_1 = m_2 \equiv \text{dom } m_1 = \text{dom } m_2 \land \forall a : A \cdot a \in \text{dom } m_1 \Rightarrow m_1(a) = m_2(a) \)

\[ m \circ n \equiv \left[ a \mapsto c | a : A, c : C, a \in \text{dom } m \land c = n(m(a)) \right] \]

\( \text{pre } \text{rng } m \subseteq \text{dom } n \)

D.4 \( \lambda \)-Calculus + Functions

D.4.1 The \( \lambda \)-Calculus Syntax

D.4.1.0.0.1 \( \lambda \)-Calculus Syntax:

\( \text{type} /\!\!\!\!\!\!\!\!* \ \text{A BNF Syntax:} /\!\!\!\!\!\!\!\!\!* /\)

\( \langle L \rangle ::= \langle V \rangle | \langle F \rangle | \langle A \rangle | ( \langle A \rangle ) \)

\( \langle V \rangle ::= /\!\!\!\!\!\!\!\!\!* \ \text{variables, i.e. identifiers} /\!\!\!\!\!\!\!\!\!\!* /\)

\( \langle F \rangle ::= \lambda \langle V \rangle \cdot \langle L \rangle \)

\( \langle A \rangle ::= ( \langle L \rangle \langle L \rangle ) \)

value /\!\!\!\!\!\!\!\!\!* /\)

\( \langle L \rangle : e, f, a, ... \)

\( \langle V \rangle : x, ... \)

\( \langle F \rangle : \lambda x \cdot e, ... \)

\( \langle A \rangle : f a, (f a), f(a), (f)(a), ... \)

D.4.2 Free and Bound Variables

D.4.2.0.0.1 Free and Bound Variables: Let \( x, y \) be variable names and \( e, f \) be \( \lambda \)-expressions.

- (V): Variable \( x \) is free in \( x \).
- (F): \( x \) is free in \( \lambda y \cdot e \) if \( x \neq y \) and \( x \) is free in \( e \).
- (A): \( x \) is free in \( f(e) \) if it is free in either \( f \) or \( e \) (i.e., also in both).

D.4.3 Substitution

In RSL, the following rules for substitution apply:
D.4.3.0.0.1 **Substitution:**

- subst\((N/x)x) ≡ N;
- subst\((N/x)a) ≡ a, for all variables \(a \neq x\);
- subst\((N/x)(P Q)) ≡ subst\((N/x)P subst\((N/x)Q));
- subst\((N/x)(\lambda y P)) ≡ \lambda y subst\((N/x)P), if x\neq y and y is not free in N or x is not free in P;
- subst\((N/x)(\lambda y P)) ≡ \lambda z subst\((N/z subst\((z/y)P))), if y \neq x and y is free in N and x is free in P (where z is not free in (N P)).

D.4.4 **\(\alpha\)-Renaming and \(\beta\)-Reduction**

D.4.4.0.0.1 **\(\alpha\) and \(\beta\) Conversions:**

- **\(\alpha\)-renaming:** \(\lambda x M\)
  If \(x, y\) are distinct variables then replacing \(x\) by \(y\) in \(\lambda x M\) results in \(\lambda y subst\((y/x)M)\). We can rename the formal parameter of a \(\lambda\)-function expression provided that no free variables of its body \(M\) thereby become bound.
- **\(\beta\)-reduction:** \((\lambda x M)(N)\)
  All free occurrences of \(x\) in \(M\) are replaced by the expression \(N\) provided that no free variables of \(N\) thereby become bound in the result. \((\lambda x M)(N) ≡ subst\((N/x)M)\)

D.4.5 **Function Signatures**

For sorts we may want to postulate some functions:

D.4.5.0.0.1 **Sorts and Function Signatures:**

- **type**
  - A, B, C
- **value**
  - obs_B: A → B,
  - obs_C: A → C,
  - gen_A: B × C → A

D.4.6 **Function Definitions**

Functions can be defined explicitly:

D.4.6.0.0.1 **Explicit Function Definitions:**

- **value**
  - f: Arguments → Result
    f(args) ≡ DValueExpr
  - g: Arguments → Result
    g(args) ≡ ValueAndStateChangeClause
  - pre P(args)

Or functions can be defined implicitly:
**D.4.6.0.0.2 Implicit Function Definitions:**

value

- \( f: \text{Arguments} \to \text{Result} \)
  - \( f(\text{args}) \) as result
  - post \( P_1(\text{args}, \text{result}) \)

- \( g: \text{Arguments} \xrightarrow{\sim} \text{Result} \)
  - \( g(\text{args}) \) as result
  - pre \( P_2(\text{args}) \)
  - post \( P_3(\text{args}, \text{result}) \)

The symbol \( \xrightarrow{\sim} \) indicates that the function is partial and thus not defined for all arguments. Partial functions should be assisted by preconditions stating the criteria for arguments to be meaningful to the function.

**D.5 Other Applicative Expressions**

**D.5.1 Simple let Expressions**

Simple (i.e., nonrecursive) \texttt{let} expressions:

**D.5.1.0.0.1 Let Expressions:**

- \( \texttt{let a = E in B(a)} \) end

is an “expanded” form of:

- \( (\lambda a. E)(B(a)) \)

**D.5.2 Recursive let Expressions**

Recursive \texttt{let} expressions are written as:

**D.5.2.0.0.1 Recursive let Expressions:**

- \( \texttt{let f = \lambda a:A \cdot E(f) in B(f,a)} \) end

is “the same” as:

- \( \texttt{let f = YF in B(f,a)} \) end

where:

- \( F \equiv \lambda g. \lambda a. (E(g)) \) and \( YF = F(YF) \)

**D.5.3 Predicative let Expressions**

Predicative \texttt{let} expressions:

**D.5.3.0.0.1 Predicative let Expressions:**

- \( \texttt{let a:A \cdot P(a) in B(a)} \) end

express the selection of a value \( a \) of type \( A \) which satisfies a predicate \( P(a) \) for evaluation in the body \( B(a) \).
D.5.4 Pattern and “Wild Card” let Expressions

Patterns and wild cards can be used:

D.5.4.0.0.1 Patterns:

\[
\begin{align*}
\text{let } \{a\} \cup s &= \text{set in ... end} \\
\text{let } \{a,\ldots\} \cup s &= \text{set in ... end} \\
\text{let } (a,b,\ldots,c) &= \text{cart in ... end} \\
\text{let } (a,\ldots,c) &= \text{cart in ... end} \\
\text{let } \langle a \rangle \hat{\ell} &= \text{list in ... end} \\
\text{let } \langle a,\ldots,b \rangle \hat{\ell} &= \text{list in ... end} \\
\text{let } [a \mapsto b] \cup m &= \text{map in ... end} \\
\text{let } [a \mapsto b,\ldots] \cup m &= \text{map in ... end}
\end{align*}
\]

D.5.5 Conditionals

Various kinds of conditional expressions are offered by RSL:

D.5.5.0.0.1 Conditionals:

\[
\begin{align*}
\text{if } b \text{ expr then } c \text{ expr else } a \text{ expr end} \\
\text{if } b \text{ expr end } \equiv /\ast \text{ same as: } */ \text{ if } b \text{ expr then } c \text{ expr else skip end} \\
\text{if } b \text{ expr_1 then } c \text{ expr_1} \\
\text{elsif } b \text{ expr_2 then } c \text{ expr_2} \\
\text{elsif } b \text{ expr_3 then } c \text{ expr_3} \\
\ldots \\
\text{elsif } b \text{ expr_n then } c \text{ expr_n end} \\
\text{case expr of} \\
\text{choice_pattern_1 } \rightarrow \text{ expr_1,} \\
\text{choice_pattern_2 } \rightarrow \text{ expr_2,} \\
\text{...} \\
\text{choice_pattern_n_or_wild_card } \rightarrow \text{ expr_n end}
\end{align*}
\]

D.5.6 Operator/Operand Expressions

D.5.6.0.0.1 Operator/Operand Expressions:

\[
\begin{align*}
\langle \text{Expr} \rangle &::= \\
&\langle \text{PrefixOp} \rangle \langle \text{Expr} \rangle \\
&| \langle \text{Expr} \rangle \langle \text{InfixOp} \rangle \langle \text{Expr} \rangle \\
&| \langle \text{Expr} \rangle \langle \text{SuffixOp} \rangle \\
&| \ldots \\
\langle \text{PrefixOp} \rangle &::= \\
&- | \sim | \cup | \cap | \text{card} | \text{len} | \text{inds} | \text{elems} | \text{hd} | \text{tl} | \text{dom} | \text{rng}
\end{align*}
\]
D.6 Imperative Constructs

D.6.1 Statements and State Changes

Often, following the RAISE method, software development starts with highly abstract-applicative constructs which, through stages of refinements, are turned into concrete and imperative constructs. Imperative constructs are thus inevitable in RSL.

D.6.1.0.0.1 Statements and State Change:

Unit value stmt:

\[
\text{stmt}: \text{Unit} \rightarrow \text{Unit} \\
\text{stmt}() 
\]

• Statements accept no arguments.
• Statement execution changes the state (of declared variables).
• Unit \(\rightarrow\) Unit designates a function from states to states.
• Statements, \text{stmt}, denote state-to-state changing functions.
• Writing () as “only” arguments to a function “means” that () is an argument of type Unit.

D.6.2 Variables and Assignment

D.6.2.0.0.1 Variables and Assignment:

0. variable \(v:\text{Type} := \text{expression}\)
1. \(v := \text{expr}\)

D.6.3 Statement Sequences and skip

Sequencing is expressed using the ‘;’ operator. \text{skip} is the empty statement having no value or side-effect.

D.6.3.0.0.1 Statement Sequences and skip:

2. \text{skip}
3. \text{stm}_1;\text{stm}_2;...;\text{stm}_n

D.6.4 Imperative Conditionals

D.6.4.0.0.1 Imperative Conditionals:

4. if \text{expr} then \text{stm}_c else \text{stm}_a end
5. case \text{e} of: \text{p}_1 \rightarrow S_1(p_1),...;\text{p}_n \rightarrow S_n(p_n) end

D.6.5 Iterative Conditionals

D.6.5.0.0.1 Iterative Conditionals:

6. while \text{expr} do \text{stm} end
7. do \text{stm} until \text{expr} end
Iterative Sequencing:

8. for e in list_expr \cdot P(b) do S(b) end

## D.7 Process Constructs

### D.7.1 Process Channels

Let A and B stand for two types of (channel) messages and i:Kdx for channel array indexes, then:

#### D.7.1.0.0.1 Process Channels:

- \texttt{channel c:A}
- \texttt{channel \{ k[i]:B \cdot i:Idx \}}
- \texttt{channel \{ k[i],...,k]:B \cdot i:Idx,j:Jdx,...,k:Kdx \}}

declare a channel, c, and a set (an array) of channels, k[i], capable of communicating values of the designated types (A and B).

### D.7.2 Process Composition

Let P and Q stand for names of process functions, i.e., of functions which express willingness to engage in input and/or output events, thereby communicating over declared channels. Let P() and Q stand for process expressions, then:

#### D.7.2.0.0.1 Process Composition:

- P ∥ Q Parallel composition
- P ⌈⌉ ⌊⌋ Q Nondeterministic external choice (either/or)
- P ⌈⌉ Q Nondeterministic internal choice (either/or)
- P -∥ Q Interlock parallel composition

express the parallel (∥) of two processes, or the nondeterministic choice between two processes: either external (⌈⌉ ⌊⌋) or internal (⌈⌉). The interlock (−∥) composition expresses that the two processes are forced to communicate only with one another, until one of them terminates.

### D.7.3 Input/Output Events

Let c, k[i] and e designate channels of type A and B, then:

#### D.7.3.0.0.1 Input/Output Events:

- c ?, k[i] ? Input
- c ! e, k[i] ! e Output

expresses the willingness of a process to engage in an event that “reads” an input, respectively “writes” an output.

### D.7.4 Process Definitions

The below signatures are just examples. They emphasise that process functions must somehow express, in their signature, via which channels they wish to engage in input and output events.
D.7.4.0.0.1 Process Definitions:

value

\[ P: \text{Unit} \rightarrow \text{in } c \text{ out } k[i] \]

\[ \text{Unit} \]

\[ Q: i:Kdx \rightarrow \text{out } c \text{ in } k[i] \text{ Unit} \]

\[ P() \equiv \ldots \text{ c } \ldots k[i] \text{ ! e } \ldots \]

\[ Q(i) \equiv \ldots k[i] \text{ ? } \ldots \text{ c } \text{ ! e } \ldots \]

The process function definitions (i.e., their bodies) express possible events.

D.8 Simple RSL Specifications

Often, we do not want to encapsulate small specifications in schemas, classes, and objects, as is often done in RSL. An RSL specification is simply a sequence of one or more types, values (including functions), variables, channels and axioms:

D.8.0.0.0.1 Simple RSL Specifications:

type

... variable

... channel

... value

... axiom

...

D.9 RSL Module Specifications

D.9.1 Modules

Modules are clusters of one or more declarations:

\[ \text{ld} = \begin{array}{l}
\text{class} \\
\hspace{1em} \text{declaration}_1 \\
\hspace{1em} \text{declaration}_2 \\
\hspace{1em} \ldots \\
\hspace{1em} \text{declaration}_n \\
\end{array} \text{ end} \]

where declarations are either

- types
- values
- axioms
- variables
- channels
- modules

By a class we mean a possibly infinite set of one or more mathematical entities satisfying the declarations.
D.9.2 Schemes

scheme Id =
  class
declaration_1
declaration_2
...
declaration_n
end

By a scheme we mean a named possibly infinite set of one or more mathematical entities satisfying the declarations.

D.9.3 Module Extension

Id = extend Id_1,Id_2,...,Id_m with
class
declaration_1
declaration_2
...
declaration_n
end

Usually we make sure that the extensions are conservative [341, 144, 100, 20, 243, 161].
Etcetera!
## INDEXES

### E.1 Definitions

Chapter 1 introduces 48 concepts and Chapters 2–10 introduce 90 definitions.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract, 3, 7</td>
<td>325</td>
</tr>
<tr>
<td>Divide and Conquer, 7</td>
<td>325</td>
</tr>
<tr>
<td>Operational, 8</td>
<td>325</td>
</tr>
<tr>
<td>Representational, 8</td>
<td>325</td>
</tr>
<tr>
<td>Algebraic Semantics, 6</td>
<td>325</td>
</tr>
<tr>
<td>Axiomatic Semantics, 6</td>
<td>325</td>
</tr>
<tr>
<td>Computer, 3</td>
<td>325</td>
</tr>
<tr>
<td>Computer Science, 3</td>
<td>325</td>
</tr>
<tr>
<td>Computing Science, 4</td>
<td>325</td>
</tr>
<tr>
<td>Conservative Extension, 7</td>
<td>325</td>
</tr>
<tr>
<td>Denotational Semantics, 6</td>
<td>325</td>
</tr>
<tr>
<td>Divide and Conquer, Abstraction, 7</td>
<td>325</td>
</tr>
<tr>
<td>Domain Engineering, 4</td>
<td>325</td>
</tr>
<tr>
<td>Domain Requirements, 4</td>
<td>325</td>
</tr>
<tr>
<td>Domain/ Machine Interface, 4</td>
<td>325</td>
</tr>
<tr>
<td>Engineering, 4</td>
<td>325</td>
</tr>
<tr>
<td>of Domain, 4</td>
<td>325</td>
</tr>
<tr>
<td>of Requirements, 5</td>
<td>325</td>
</tr>
<tr>
<td>of Software, 6</td>
<td>325</td>
</tr>
<tr>
<td>Epistemology, 4</td>
<td>325</td>
</tr>
<tr>
<td>Formal Method:, 4</td>
<td>325</td>
</tr>
<tr>
<td>Formalisation, Narration, 8</td>
<td>325</td>
</tr>
<tr>
<td>Hardware, 4</td>
<td>325</td>
</tr>
<tr>
<td>Informatics, 8</td>
<td>325</td>
</tr>
<tr>
<td>Information Technology, 8</td>
<td>325</td>
</tr>
<tr>
<td>Intentional Pull, 8</td>
<td>325</td>
</tr>
<tr>
<td>Interface of Domain/ Machine, 4</td>
<td>325</td>
</tr>
<tr>
<td>Invariant, 8</td>
<td>325</td>
</tr>
<tr>
<td>Language, 4</td>
<td>325</td>
</tr>
<tr>
<td>Linguistics, 4</td>
<td>325</td>
</tr>
<tr>
<td>Machine, 4</td>
<td>325</td>
</tr>
<tr>
<td>Machine Requirements, 4</td>
<td>325</td>
</tr>
<tr>
<td>Mathematics, 4</td>
<td>325</td>
</tr>
<tr>
<td>Mereology, 5</td>
<td>325</td>
</tr>
<tr>
<td>Metaphysics, 5</td>
<td>325</td>
</tr>
<tr>
<td>Method, 5, 7</td>
<td>325</td>
</tr>
<tr>
<td>Principle, 7</td>
<td>325</td>
</tr>
<tr>
<td>Technique, 7</td>
<td>325</td>
</tr>
<tr>
<td>Tool, 7</td>
<td>325</td>
</tr>
<tr>
<td>Methodology, 5</td>
<td>325</td>
</tr>
<tr>
<td>Model, 5</td>
<td>325</td>
</tr>
<tr>
<td>Modelling, 5</td>
<td>325</td>
</tr>
<tr>
<td>Narration Formalisation, 8</td>
<td>325</td>
</tr>
<tr>
<td>Nondeterminism, 8</td>
<td>325</td>
</tr>
<tr>
<td>Ontology, 5</td>
<td>325</td>
</tr>
<tr>
<td>Operational Abstraction, 8</td>
<td>325</td>
</tr>
<tr>
<td>Philosophy, 5</td>
<td>325</td>
</tr>
</tbody>
</table>
Pragmatics, 5
Principle
of a Method, 7
Pull, Intentional, 8
Refinement, 8
Representational Abstraction, 8
Requirements, 5
Engineering, 5
Prescription, 6
Specification, 6
Science, 6
of Computers, 3
Semiotics, 6
and Syntax, 8
Software, 6
Design, 6
Development, 6
Engineering, 6
Syntax, 7
and Semantics, 8
Names, 8
Taxonomy, 7
Technique
of a Method, 7
Technology, 7
Technology, Information, 8
Tool
of a Method, 7
Triptych, 7

Chapters 2–10 Definitions
1. Transcendental, I, 12
10. Axioms and Axiom System, 18
11. Proof, 18
12. Interpretation, 18
13. Satisfiability, 18
14. Validity, 18
15. Model, 18
16. Type, 19
17. Sort, 19
18. Function, 20
19. Signature, 20
2. Transcendental Deduction, I, 12
20. Total Function, 20
21. Partial Function, 20
22. Predicate, 20
23. Indefinite Space, 22
24. Definite Space, 22
25. Indefinite Time, 25
26. Definite Time, 25
27. Domain, 39
28. Domain Description, 40
29. External Quality, 42
3. Necessarily True Assertions, 13
30. Entity, 43
31. Endurant, 47
32. Perdurant, 47
33. Discrete Endurant, 48
34. Continuous Endurant, 48
35. Compound Endurants, 49
36. Root, 49
37. Sibling, 49
38. Physical Parts, 50
39. Endurant Structure, 50
4. Possibly True Assertions, 13
40. Living Species, I, 51
41. Natural Parts, 51
42. Man-made Parts: Artefacts, 51
43. Compound Structure, 52
44. Set Structure, 53
45. Living Species, II, 53
46. Animal, 54
47. Human, 54
48. Material, 54
49. Atomic Part, 56
5. Space, 14
50. Composite Part, 56
51. Part-Materials Conjoin, 57
52. Material-Parts Conjoin, 58
53. State, 61
54. Formal Context, 76
55. Qualities Common to a Set of Entities, 76
56. Entities Common to a Set of Qualities, 76
57. Formal Concept, 76
58. Mereology, 89
59. Fixed Mereology, 92
6. Time, 14
60. Varying Mereology, 92
61. Transcendental, II, 123
62. Transcendental Deduction, II, 123
63. Transcendentiality, 123
64. Actor, 127
65. Discrete Action, 127
66. Event, 127
67. Discrete Behaviour, 127
68. Function Signature, 136
69. Function Type Expression, 136
7. Logic, 17
70. Requirements (I), 208
71. Requirements (II), 208
72. Machine, 209
73. Requirements (III), 209
74. Problem, Solution and Objective Sketch, 209
75. System Requirements, 210
76. User and External Equipment Requirements, 210
77. Assumption and Design Requirements, 211
E.1 Definitions

78. Verification Paradigm, 211
79. Domain Requirements Prescription, 212
8. Mathematical Logic, 18
80. Domain Projection, 213
81. Domain Instantiation, 216
82. Determination, 219
83. Extension, 221
84. Endurant Extension, 221
85. Requirements Fitting, 228
86. Requirements Harmonisation, 228
87. Derived Perdurant, 235
88. Derived Action, 235
89. Derived Event, 236
9. Inference Rules, 18
90. Domain Specific Language, 259

Miscellaneous “in-line” Definitions, 17
actively mobile, 115
animals
  intentionality, 111
animate entities
  intentionality, 111
attributes
  Attributes
    Fuzzy, 109–110
Attributes
  of Conjoins, 108–109

Behaviour
  Definitions, 140–150
  Signatures, 140–146

Business
  Process
    Engineering, 257
    Re-engineering, 257

causality
  intentionality, 111
composite
  endurant, 55
compound
  endurant, 49
confusion
  of endurants, 116
conjoin
  dispose, 93
  endurant, 55
  flow net, 93
material-parts, 55
  part-materials, 55
part-parts, 55
  pipe, 93
  process, 93
  pump, 93
  supply, 93
  transport, 93
  treatment, 93
  valve, 93
Conjoin
  Attributes, 108–109
CSP
  event, 154
dispose
  conjoin, 93
domain
  event, 155
Duration Formula, DC, 32
Duration Term, DC, 32
Duration, DC, 32
endurant
  composite, 55
  compound, 49
  confusion, 116
  conjoin, 55
  junk, 116
  root, 49
  sibling, 49
event
  domain, 155
  event
    CSP, 154
flow
  net of conjoins, 93
Fuzzy
  Attribute
    types, 110
    values, 110
  Attributes, 109–110
  Sets and Fuzzy Logic, 110
Galois
  Connection, 114–115
  intentionality, 115
  Galois connection, 114
humans
  consciousness and learning
    intentionality, 111
Identity, 34
intentional
  pull, 111
humans
  consciousness and learning, 111
intentional
  pull, 111
E.2 Examples

There are 128 examples.

A Casually Described Bank Script, 173
A Conjoin Canal Lock, 143
A Formally Described Bank Script, 174
A Human Behaviour Mortgage Calculation, 189
A Road Transport Domain, 41
   I: Composite, 68
   II: Structure, 69
   III: Part-Parts, 72
A Road Transport Domain, I: Composite, 68

A Road Transport Domain, II: Structure, 69
A Road Transport Domain, III: Part-Parts, 72
A Road Transport System: Abstract External Qualities, 42
A Road Transport System: Manifest External Qualities, 42
A Rough Sketch Domain Endurant Description, 50
Action and Event Attributes, 103
Actors and Actions, 181
<table>
<thead>
<tr>
<th>Example</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bank - or Programming - Staff Behaviour</td>
<td>188</td>
</tr>
<tr>
<td>Buses and Bus Companies, 129</td>
<td></td>
</tr>
<tr>
<td>Civil Engineering: Consultants and Contractors, 115</td>
<td></td>
</tr>
<tr>
<td>Composite Automobile Parts, 56</td>
<td></td>
</tr>
<tr>
<td>Constants and States, 62</td>
<td></td>
</tr>
<tr>
<td>Container Terminal Port, 116</td>
<td></td>
</tr>
<tr>
<td>Container Terminal Port, 116</td>
<td></td>
</tr>
<tr>
<td>Credit Card Shopping System, 116</td>
<td></td>
</tr>
<tr>
<td>Critical Resource Sharing, 130</td>
<td></td>
</tr>
<tr>
<td>Digital Media, 177</td>
<td></td>
</tr>
<tr>
<td>Discrete Endurants, 48</td>
<td></td>
</tr>
<tr>
<td>Document Artefactual Attributes, 108</td>
<td></td>
</tr>
<tr>
<td>Documents, 178</td>
<td></td>
</tr>
<tr>
<td>Domain Events, 155</td>
<td></td>
</tr>
<tr>
<td>Endurants, 47</td>
<td></td>
</tr>
<tr>
<td>Fixed and Varying Mereology, 92</td>
<td></td>
</tr>
<tr>
<td>Further Behaviours of a Road Transport System, 147</td>
<td></td>
</tr>
<tr>
<td>General Hospital System, 116</td>
<td></td>
</tr>
<tr>
<td>Hard Real-Time Models Expressed in “Ordinary” RSL Logic, 29</td>
<td></td>
</tr>
<tr>
<td>Health-care, 178</td>
<td></td>
</tr>
<tr>
<td>Inert Attribute, 96</td>
<td></td>
</tr>
<tr>
<td>Inescapable Meaning Assignment, Formalisation, 12</td>
<td></td>
</tr>
<tr>
<td>Inescapable Meaning Assignment, Narrative, 11</td>
<td></td>
</tr>
<tr>
<td>Initial System, 151</td>
<td></td>
</tr>
<tr>
<td>Intentional Pull, I, 112</td>
<td></td>
</tr>
<tr>
<td>Intentional Pull, II, 114</td>
<td></td>
</tr>
<tr>
<td>Intents, 114</td>
<td></td>
</tr>
<tr>
<td>Intrinsics of Switches, 165</td>
<td></td>
</tr>
<tr>
<td>Invariance of Road Net Traffic States, 98</td>
<td></td>
</tr>
<tr>
<td>Invariance of Road Nets, 91</td>
<td></td>
</tr>
<tr>
<td>LEGO Blocks, 114</td>
<td></td>
</tr>
<tr>
<td>Licensors and Licensees, 180</td>
<td></td>
</tr>
<tr>
<td>Material of an Airspace, 64</td>
<td></td>
</tr>
<tr>
<td>Materials, 49</td>
<td></td>
</tr>
<tr>
<td>Materials of Wastewater Management Plant, 65</td>
<td></td>
</tr>
<tr>
<td>Mereology of a Road Net, 91</td>
<td></td>
</tr>
<tr>
<td>Natural and Man-made Materials, 55</td>
<td></td>
</tr>
<tr>
<td>Natural Parts: River Systems, 51</td>
<td></td>
</tr>
<tr>
<td>Part-Material Conjoins: Canals with Locks, I, 58</td>
<td></td>
</tr>
<tr>
<td>Part-Material Conjoins: Pipelines, I, 58</td>
<td></td>
</tr>
<tr>
<td>Part-Materials Conjoins: Waste Management, I, 58</td>
<td></td>
</tr>
<tr>
<td>Part-Parts Conjoin: Container Terminal Ports, 59</td>
<td></td>
</tr>
<tr>
<td>Parts of a Road Net, 65</td>
<td></td>
</tr>
<tr>
<td>Parts of an Airspace, 64</td>
<td></td>
</tr>
<tr>
<td>Passenger and Goods Transport, 179</td>
<td></td>
</tr>
<tr>
<td>Perdurants, 47</td>
<td></td>
</tr>
<tr>
<td>Physical Parts, 51</td>
<td></td>
</tr>
<tr>
<td>Pipeline Attributes, 98</td>
<td></td>
</tr>
<tr>
<td>Pipeline Mereology, 94</td>
<td></td>
</tr>
<tr>
<td>Pipeline Parts and Material, 71</td>
<td></td>
</tr>
<tr>
<td>Pipeline Unique Identifiers, 89</td>
<td></td>
</tr>
<tr>
<td>Plants, 53</td>
<td></td>
</tr>
<tr>
<td>Possible Consequences of a Road Net Mereology, 92</td>
<td></td>
</tr>
<tr>
<td>Probabilistic Rail Switch Unit State Transitions, 168</td>
<td></td>
</tr>
<tr>
<td>Programmable Attribute, 97</td>
<td></td>
</tr>
<tr>
<td>Railway Management and Organisation: Train Monitoring, II, 184</td>
<td></td>
</tr>
<tr>
<td>Railway Net Intrinsics, 164</td>
<td></td>
</tr>
<tr>
<td>Railway Optical Gates, 170</td>
<td></td>
</tr>
<tr>
<td>Railway Support Technology, 167</td>
<td></td>
</tr>
<tr>
<td>Reactive Attributes, 97</td>
<td></td>
</tr>
<tr>
<td>Requirements:</td>
<td></td>
</tr>
<tr>
<td>Domain Requirements,</td>
<td></td>
</tr>
<tr>
<td>Derived Action – Tracing Vehicles, 235</td>
<td></td>
</tr>
<tr>
<td>Derived Event, Current Maximum Flow, 236</td>
<td></td>
</tr>
<tr>
<td>Determination – Toll-roads, 220</td>
<td></td>
</tr>
<tr>
<td>Endurant Extension 1/2, 221</td>
<td></td>
</tr>
<tr>
<td>Endurant Extension 2/2, 223</td>
<td></td>
</tr>
<tr>
<td>Fitting, 229</td>
<td></td>
</tr>
<tr>
<td>Instantiation – Road Net 1/2, 216</td>
<td></td>
</tr>
<tr>
<td>Instantiation of Road Net, Abstraction, 219</td>
<td></td>
</tr>
<tr>
<td>Instantiation, Road Net 2/2, 217</td>
<td></td>
</tr>
<tr>
<td>Projection, 213</td>
<td></td>
</tr>
<tr>
<td>Projection – A Narrative Sketch, 213</td>
<td></td>
</tr>
<tr>
<td>Interface Requirements,</td>
<td></td>
</tr>
<tr>
<td>Projected Extensions, 230</td>
<td></td>
</tr>
<tr>
<td>Shared Behaviours, 233</td>
<td></td>
</tr>
<tr>
<td>Shared Endurant Initialisation, 231</td>
<td></td>
</tr>
</tbody>
</table>

July 6, 2020, 10:03, A Foundation for Software Development
INDEXES

Shared Endurants, 230
Road Pricing, 210
Design Assumptions, 212
Design Requirements, 212
Toll-Gate System, Design Assumptions, 212
Toll-Gate System, Design Requirements, 212
User and External Equipment, Narrative, 210
Requirements: Sketch of Objectives, 209
Road Net, 106
Road Net Artefactual Attributes, 108
Road Net Attributes, 98
Road Transport Behaviour Signatures, 144
Road Transport: Further Attributes, 98
Road Transport: Unique Identifier Auxiliary Functions, 87
Shopping System
Credit Card, 116
Soft Real-Time Models Expressed in “Ordinary” RSL Logic, 29

Some Transcendental Deductions, 123
State Values versus State Variables, 126
Static Attributes, 96
Strategic, Tactical and Operations Management, 186
Structures versus Composites, 52
Temporal Notions of Endurants, 25
Terminal Port
Container, 116
The Shipping Lanes, 59
Traffic Signals, 168
Train Monitoring, I, 184
Trains Along Lines, 171
Trains at Stations, 171
Transcendentity, 123
Transport System Structures, 52
Unique Identifier Constants, 88
Unique Identifiers, 87
Uniqueness of Road Net Identifiers, 89
Universes of Discourse, 40

E.3 Method Hints

We have made 9 explicit method hints.

1 Select Domain of Interest, 41
2 External Qualities, 42
3 ‘What can be Described’, 43
4 Initial Focus is on Endurants, 47
5 Discrete versus Continuous, 49
6 From Analysis to Description, 60
7 Domain State, 62
8 Put New Endurants on Hold, 68
9 Sequential Analysis & Description of Internal Qualities, 85

E.4 Analysis Predicate Prompts

There are 21 analysis predicates.

is. animal, 54
is. artefactual. composite, 56
is. artefactual. atomic, 56
is. artefact, 51
is. atomic, 56
is. composite, 56
is. compound. structure, 52
is. conjoin, 57
is. continuous, 49
is. discrete, 48
is. endurant, 47
is. entity, 43
is. human, 54
is. living. species, 51
is. material. parts. conjoin, 59

is. material, 49
is. natural. atomic, 56
is. natural. composite, 56
is. natural. part, 51
is. part. materials. conjoin, 57
is. part. parts. conjoin, 59
is. perdurant, 48
is. physical. part, 50
is. plant, 53
is. set. structure, 53
is. structure, 50
has. monitorable. attributes, 126
is. physical. attribute, 205, 107
E.5 Analysis Function Prompts

There are 19 analysis functions.

- `analyse_attribute_types`, 95
- `analyse_compound_structure_sorts`, 64
- `analyse_endurant_sorts`, 60
- `analyse_intentionality`, 112
- `analyse_material_parts_material`, 64
- `analyse_material_parts_parts`, 64
- `analyse_material`, 65
- `analyse_part_materials_materials`, 65
- `analyse_part_materials_part`, 65
- `analyse_part_parts`, 65
- `analyse_set_structure_sort`, 64
- `calc_parts`, 104, 62
- `calculate_all_unique_identifiers`, 87
- `type_name`, `type_of`, 66–68, 70–73, 87
- `analyse_abs_phys_attr`, 206, 107
- `analyse_conc_phys_attr`, 207, 107
- `is_`, 87
- `is_`, 70–73
- `obs_AbsPhysUnit_Attr`, 202, 107
- `obs_ConcPhysUnit_Attr`, 203, 107
- `obs_PhysScale_Attr`, 204, 107
- `possible_variable_declaration`, 126
- `type_name`, 66–68
- `type_of`, 66–68

E.6 Attribute Categories

There are 9 attribute categories.

- `is_active_attribute`, 97
- `is_autonomous_attribute`, 97
- `is_biddable_attribute`, 97
- `is_dynamic_attribute`, 96
- `is_inert_attribute`, 96
- `is_monitorable_only_attribute`, 98
- `is_programmable_attribute`, 97
- `is_reactive_attribute`, 96
- `is_static_attribute`, 96

E.7 Perdurant Calculations

- `calc_all_chn_dcls`, Item 243a, 139
- `calc_chn_refs`, Item 242a, 139
- `calc_chn_refs`, Item 241, 139
- `declaring_all_monitorable_variables`, Item 229, 126
- `moni_attr_types`, Item 183, 101
- `moni_attr_vals`, Item 186, 101
- `prgr_attr_types`, Item 184, 101
- `prgr_attr_vals`, Item 188, 101
- `stat_attr_types`, Item 182, 101
- `stat_attr_vals`, Item 186, 101
- Translate Endurant, 140

E.8 Description Prompts

There are 11 description prompts.

- `describe_attributes`, 95
- `describe_compound_structure_sorts`, 67
- `describe_endurant_sorts`, 66
- `describe_material_parts_sorts`, 70
- `describe_material_sort`, 73
- `describe_mereology`, 90
E.9 Endurant to Perdurant Translation Schemas

There are 9 perdurant schemas.

possible_variable_declaration, 126
Translate Atomic, 141
Translate Composite, 142
Translate Conjoin, 142
Translate Endurant, 126
Translate Material-Parts, 143
Translate Part-Materials, 143
Translate Part-Parts, 144
Translate Structure, 142

E.10 RSL Symbols

Literals, 314–323
Unit, 323
chaos, 314, 315
false, 308, 310
true, 308, 310

Arithmetic Constructs, 310
a * a, 310
a + a, 310
a = a, 310
a ≥ a, 310
a > a, 310
a ≤ a, 310
a < a, 310
a ≠ a, 310
□, 310
⇒, 310
=, 310
#, 310
~, 310
∨, 310
∧, 310

Cartesian Constructs, 311, 314
(e_1, e_2, ..., e_n), 311

Combinators, 319–322
... elsif ..., 320
case b_of pa_1 → c_1, ..., pa_n → c_n end, 320, 321
do stmt until be end, 321
for e in list_expr • P(b) do stmt(e) end, 322
if b, then c_1 else c_n end, 320, 321
let a:A • P(a) in c end, 319

Logic Constructs, 309–310
b ∨ b, 310
∀ a:A • P(a), 310
∃ b: A • P(b), 310
∃ a:A • P(a), 310
~ b, 310
false, 308, 310
true, 308, 310
b_i \Rightarrow b_j , 310
b_i \land b_j , 310

Map Constructs, 312, 316–317
m_i \setminus m_j , 316
m_i \cap m_j , 316
m_i / m_j , 316
dom m , 316
rng m , 316
m_i \not\leftrightarrow m_j , 316
m_i = m_j , 316
m_i \cup m_j , 316
m(e) , 316
[u_1 \mapsto v_1, u_2 \mapsto v_2, ..., u_n \mapsto v_n] , 312
[F(e) \mapsto G(m(e)) | e \in \text{dom} m \land P(e)] , 312

Modules, 323–324
\text{Id extend Id}_1, \text{Id}_2, ..., \text{Id}_m \text{ with class } ... \text{ end}, 324
class ... \text{ end}, 323
\text{scheme Id = class } ... \text{ end}, 324
module, 323

Process Constructs, 322–323
channel c: T , 322
channel \{k[i]: T\textbull;Idx\}, 322
c! e , 322
c? , 322
k[i]! e , 322
k[i]? , 322
p_i! p_j , 322
p_i| p_j , 322
p_i\parallel p_j , 322
p_i\llp p_j , 322
P: \text{Unit } \rightarrow \text{in c out k[i] Unit} , 323
Q: i: Idx \rightarrow \text{out c in k[i] Unit}, 323

Set Constructs, 310–314
\cap \{s_1, s_2, ..., s_n\} , 312
\cup \{s_1, s_2, ..., s_n\} , 312
card s , 312
e \in s , 312
e \not\in s , 312
s_i = s_j , 312
s_i \cap s_j , 312
s_i \cup s_j , 312
s_i \subseteq s_j , 312
s_i \not\subseteq s_j , 312
s_i \setminus s_j , 312
\{\} , 310
\{e_1, e_2, ..., e_n\} , 310
\{Q(a) | a: A \land a \in s \land P(a)\} , 311

Type Expressions, 307–308
(T_1 \times T_2 \times ... \times T_n) , 308
\text{Bool} , 307
\text{Char} , 307
\text{Int} , 307
\text{Nat} , 307
\text{Real} , 307
\text{Text} , 307
\text{Unit} , 321
mk_id(s_1:T_1, s_2:T_2, ..., s_n:T_n) , 308
s_1:T_1 s_2:T_2 ... s_n:T_n , 308
T^\ast , 308
T^0 , 308
T_1 \times T_2 \times ... \times T_n , 308
T_1 | T_2 | ... | T_1 | T_n , 308
T_i \not\leftrightarrow T_j , 308
T_i \leftrightarrow T_j , 308
T_i \rightarrow T_j , 308
T\text{-infset} , 308
T\text{-set} , 308

Type Definitions, 308–309
\text{T = Type\textbull;Expr} , 308
\text{T=} [[v:T\bullet P(v)]] , 309
T\Rightarrow TE_1 | TE_2 | ... | TE_n , 309