Domain Science & Engineering
A Review of 10 Years Work and a Laudatio
The ZCC Fest, 20 October 2017, Changsha, China

Dines Bjørner

Technical University of Denmark, DK-2800 Kgs.Lyngby, Denmark
Fredsvej 11, DK-2840 Holte, Danmark
`bjorner@gmail.com`, www.imm.dtu.dk/~db

Abstract. A personal account is given of my scientific work since I retired 10 years ago. This work centers around a new dimension to computing science: that of domain science & engineering. By a *domain* we shall understand a *rationally describable* segment of a *human assisted* reality, i.e., of the world, its *physical parts*, and *living species*. These are *endurants* ("still"), existing in space, as well as *perdurants* ("alive"), existing also in time. Emphasis is placed on *"human-assistedness"*, that is, that there is *at least one (man-made) artifact* and that *humans* are a primary cause for change of endurant *states* as well as perdurant *behaviours*. Section 8 brings my laudatio.

1 Introduction

I survey recent work in the area of *domain science & engineering*[1].

A strict interpretation of the *triptych* of software engineering dogma suggests that software development "ideally" proceeds in three phases:

- First a phase of *domain engineering* in which an analysis of the application domain leads to a description of that domain.
- Then a phase of *requirements engineering* in which an analysis of the domain description leads to a prescription of requirements to software for that domain.
- And, finally, a phase of *software design* in which an analysis of the requirements prescription leads to software for that domain.

[1] It is appropriate, at this point, to state that my use of the term 'domain' is not related to that of *Domains and Processes* such as in the *Proceedings of 1st International Symposium on Domain Theory, Shanghai, China, October 1999, eds.: Klaus Keimel, Zhang Guo-Qiang, Liu Ying-Ming and Chen Yi-Chang. Springer Science + Business Media, New York, 2001.*

We see *domain science & engineering* as a discipline that *need not be justified as a precursor to requirements engineering. Just as physicists study nature, irrespective of engineering, so we can study manifest domains irrespective of computing.*

1.1 Recent Papers and Reports

Over the last decade I have iterated a number of investigations of aspects of this *triptych* dogma. This has resulted in a number of papers (and revised reports):

– Manifest Domains: Analysis & Description (2018, 2014) [29, 35]
– Domain Facets: Analysis & Description (2018. 2008) [31, 12]
– From Domains to Requirements (2018, 2008) [25, 8]
– Formal Models of Processes and Prompts (2014,2017) [23, 20]
– To Every Domain Mereology a CSP Expression (2017, 2009) [33, 10]
– A Philosophy of Domain Science & Engineering (2018) [30]

[30], a report, is the most recent.

1.2 Recent Experiments

Applications of the domain science and engineering outlined in [29]–[8] are exemplified in reports and papers on experimental domain analysis & description. Examples are:

– *Urban Planning* [41],
– *Documents* [28],
– *Credit Cards* [22],
– *Weather Information Systems* [26],
– *The Tokyo Stock Exchange* [34],
– *Pipelines* [18],
– *Road Transportation* [19],
– *Web/Transaction-based Software* [14],
– *"The Market"* [4],
– *Container [Shipping] Lines* [7],
– *Railway Systems* [3, 37, 5, 51, 56].

1.3 My Emphasis on Software Systems

An emphasis in my work has been on research into and experiments with application areas that required seemingly large scale software. Not on tiny, beautiful, essential data structures and algorithms.

I first worked on the proper application of formal methods in software engineering at the *IBM Vienna Laboratory* in the early 1970s. That was to the formalisation of the semantics of IBMs leading programming language then, *PL/I*, and to a systematic development of a compiler for that language. The latter never transpired.

Instead I got the chance to formulate the stages of development of a compiler from a denotational semantics description to so-called "running code" [2, 1977]. That led, from 1978 onward, to two MSc students and a colleague and I working on a formal description of the *CCITT Communications High Level Language, CHILL* and its compiler [1, 46]. And that led, in 1980, to five MSc students of ours producing a formal description of a semantics for the *US DoD Ada programming language, Ada* [40]. And that led to the formation of *Dansk Datamatik Center* [38] which embarked on the CHILL and Ada compiler developments [42, 50]. *To my knowledge that project which was on time, at budget, and with a history of less that 3% cost of original budget for subsequent error correction over the first 20 years of use of that compiler was a first, large, successful example of the systematic use of formal methods in large scale (42 man years) software development.*

1.4 How Did We Get to Domain Science & Engineering ?

So that is how we came from the semantics of programming languages to the semantics of human-centered, manifest application domain software development. Programming language semantics has to do with the meaning of abstract concepts such as programs, procedures, expressions, statements, GOTOs, labels, etc. Domain semantics, for manifest domains, in so far as we can narrate and formalize it, or them, must capture some "meanings" of the manifest objects that we can touch and see, of the actions we perform on them, and of the sentences by means of which we talk about those phenomena in the domain.

1.5 Preliminaries

We need formulate a few characterisations.

Method & Methodology: By a *method* I understand a set of principles for selecting and applying techniques and tools for constructing a manifest or an abstract artifact.

By *methodology* I understand the study and knowledge of methods.

My contributions over the years have contributed to methods for software design and, now, for the last many years, methods for domain analysis & description.

In my many experiments with domain analysis & description, cf. Sect. 5 on Page 22, I have found that I often let a so-called "streak of creativity" enter my analysis & description – and, as a result I get stuck in my work. Then I recall, ah!, but there are these principles, techniques and tools for analysis & description, and once I apply them, "strictly", i.e., methodically, I am back on the track, and, in my view, a more beautiful description emerges!

Computer & Computing Sciences: By *computer science* I understand the study and knowledge about the things that can exist inside computing devices.

By *computing science* I understand the study and knowledge about how to construct the things that can exist inside computing devices. Computing science is also often referred to as *programming methodology. My work is almost exclusively in the area of computing science.*

A Triptych of Informatics: Before software can be designed we must have a firm grasp on its/their requirements. Before requirements can be prescribed we must have a firm grasp on their basis: the domain. We therefore see informatics as consisting of

- *domain science & engineering*,
- *requirements science & engineering*, and
- *programming methodology*.

This paper contributes to the establishment of *domain science & engineering*, while hinting that *requirements science & engineering* can benefit from the relation between the two [25, 8]. *How much of a domain must we analyse & describe* before we attempt the second and third phases of the triptych ?. When this question is raised, after a talk of mine over the subject, and by a colleague researcher & scientist I usually reply: *As large a domain as possible!* This reply is often met by this *comment* (from the audience) *Oh ! No, that is not reasonable!* To me that comment shows either or both of: the questioner was not asking as a researcher/scientist, but as an engineer. Yes, an engineer needs only analyse & describe up to and slightly beyond the "border" of the domain-of-interest for a current software development – but a researcher cum scientist is, of course, interested not only in a possible requirements engineering phase beyond domain engineering, but is also curious about the larger context of the domain, in possibly establishing a proper domain theory, etc.

1.6 The Papers

IM^{2}HO I consider the first of the papers reviewed, [29], *my most important paper.* It was conceived of last[2], after publication of three of the other papers [12, 8, 16]. Experimental evidence then necessitated extensive revisions to these other papers, resulting in [31, 25, 24].

1.7 Structure of This Paper

Section 2 reviews [29, *Analysis & Description Prompts*], and Sect. 3 reviews related science and methodology papers. [31, *Domain Facets*] (Sect. 3.1), [25, *From Domains to Requirements*] (Sect. 3.2), [23, *An Analysis & Description Process Model*] (Sect. 3.3), and [33, *From Mereologies to Lambda-Expressions*] (Sect. 3.4), Finally, Sect. 4 briefly reviews [30, *A Philosophy Basis*] *work-in-progress*.

2 Manifest Domains: Analysis & Description [29]

This work grew out of many years of search for principles, techniques and tools for systematically analyzing and describing manifest domains. By a manifest domain we shall understand a domain whose entities we can observe and whose endurants we can touch!

2.1 A Domain Ontology

Parts, Components and Materials: The result became a calculus of analysis and description prompts[3]. These prompts are tools that the domain analyser & describer uses. The domain analyser & describer is in the domain, sees it, can touch it, and then applies the prompts, in some orderly fashion, to what is being observed. So, on one hand, there is the necessarily informal domain, and, on the other hand, there are the seemingly formal prompts and the *"suggestions for something to be said"*, i.e., written down: narrated and formalised. Figure 1 on the next page suggests a number of *analysis* and *description* prompts. The domain analyser & describer is "positioned" at the top, the "root". If what is observed can be conceived and described then it *is an entity*. If it can be described as a "complete thing" at no matter which given snapshot of time then it *is an endurant*. If it is an entity but for which only a fragment exists if we look

[2] Publication [13, 15] is a predecessor of [35] which is then a predecessor of [29].

[3] Prompt, as a verb: to move or induce to action; to occasion or incite; inspire; to assist (a person speaking) by *"suggesting something to be said"*.

at or touch them at any given snapshot in time, then it *is a perdurant*. Endurants are either *discrete* or *continuous*. With discrete endurants we can choose to associate, or to not associate *mereologies*[4]. If we do we shall refer to them as *parts*, else we shall call them *components*. The continuous endurants we shall also refer to as *(gaseous or liquid) materials*. *Parts* are

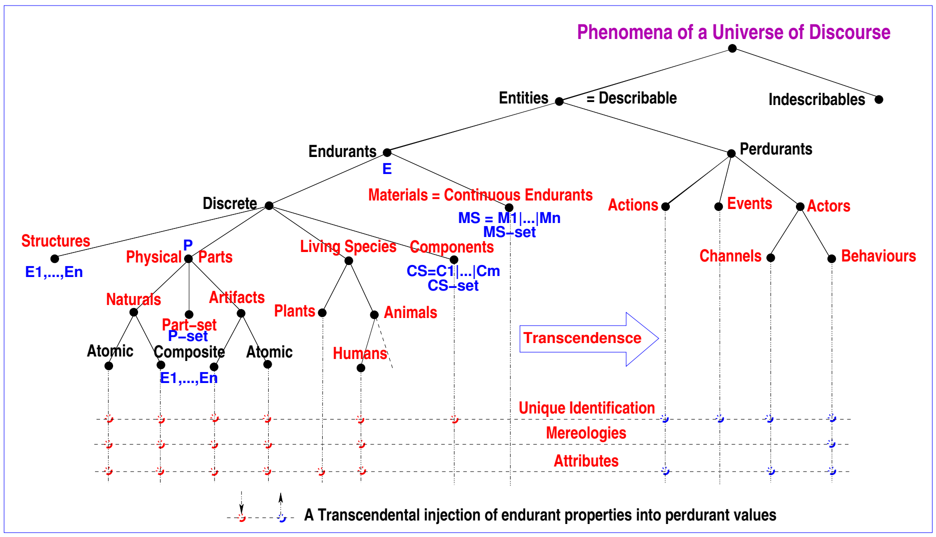

Fig. 1. Domain Ontology

either *atomic* or *composite* and all parts have *unique identifiers*, *mereology* and *attributes*. Atomic parts *may have* one or more *components* and/or one or more *materials*

If the observed part, p:P, is composite then we can observe the part sorts, $P_1, P_2, ..., P_m$ of p: *observe_part_sorts*(p) which yields the informal and formal description: **Narrative:** ... **Formal: type** $P_1, P_2, ..., P_m$, **value** obs_P$_i$: $P \rightarrow P_i$, repeated for all m part sorts P_is" !

Part sorts may have a concrete type: *has_concrete_type*(p) in which case *observe_concrete_part_type*(p) yields **Narrative:** ... **Formal: type:** $T = P$**-set**, **value** obs_T: $P \rightarrow K$**-set** – where K**-set** is one of the concrete type forms, and where K is some sort.

Materials have types (i.e., are of sorts): M_i. Observing the (one) material, of type M, of an endurant e of sort E is expressed as *obs_materials*(e)

[4] — 'mereology' will be explained next

which yields some narrative and some formal *description text:* **Narrative:** ... **Formal: type** M **value** obs_M: $E \rightarrow M$. The narrative text (...) narrates what the formal text expresses[5].

Components, i.e., discrete endurants for whom we do not consider possible mereologies or attributes, can be observed from materials, $m : M$, or are just observed of discrete endurants, $e : E$: *obs_components*(em) which yields the informal and formal description: **Narrative:** ... **Formal: type:** $C_1, C_2, ..., C_n$ **value** obs_C$_i$: $(E|M) \rightarrow C_i$ repeated for all n component sorts Cs" to the formal text !

• • •

The above is a pedagogic simplification. As shown in Fig. 1 on the facing page there are not only parts. There are also *living species: plants* and *animals*, including *humans*. And, because there are humans in the domains, parts and materials are either *natural* or *artifacts* (man-made). Humans create artifacts, usually with an *intent*. Humans have *intents*, and artifacts "possess" *intents*. Intents are like *attributes*, see below.

• • •

We have just summarised the analysis and description aspects of endurants in *extension* (their "form"). We now summarise the analysis and description aspects of endurants in *intension* (their "contents"). There are three kinds of intensional *qualities* associated with parts, two with components, and one with materials. Parts and components, by definition, have *unique identifiers*; parts have *mereologies*, and all endurants have *attributes*.

Unique identifiers: Unique identifiers are further undefined tokens that uniquely identify parts and components. The description language observer *uid_P*, when applied to parts p:P yields the unique identifier, π:Π, of p. The *observe_part_sorts(p)* invocation yields the description text: ... [added to the narrative and] **type** $\Pi_1, \Pi_2, ..., \Pi_m$; **value** uid_Π_i : $P_i \rightarrow \Pi_i$, repeated for all m part sorts P_is and added to the formalisation.

Mereology: *Mereology is the study and knowledge of parts and part relations.* The mereology of a part is an expression over the unique identifiers

[5] – not how it expresses it, as, here, in the `RAISE [44]` Specification Language, `RSL` [43].

of the (other) parts with which it is related, hence *mereo_P:* $P \rightarrow \mathcal{E}(\Pi_j$,-..., $\Pi_k)$, $\mathcal{E}(\Pi_j, ..., \Pi_k)$ is a type expression. The *observe_part_sorts*(p) invocation yields the description text: ... [added to the narrative and] **value** mereo_$P_i : P_i \rightarrow \mathcal{E}_i(\Pi_{i_j}, ..., \Pi_{i_k})$ [added to the formalisation]
Example: The mereologies, (i, o), of pipe units in a pipeline system thus express, for each kind of pipe unit, whether it is a well, a linear pipe, a fork, a join, a pump, a valve, or a sink, the identities of the zero, one or two pipe units that it is "connected" to on the input, i, respectively the output, o, side: for well $(0, 1)$, for pipe $(1, 1)$, for fork $(1, 2)$, for join $(2, 1)$, for valve $(1, 1)$, for pump $(1, 1)$, for sink $(1, 0)$ units

Attributes: Attributes are the remaining qualities of endurants. The analysis prompt *obs_attributes* applied to an endurant yields a set of type names, $A_1, A_2, ..., A_t$, of attributes. They imply the additional description text: **Narrative:** ... **Formal: type** $A_1, A_2, ..., A_t$ **value** attr_A$_i$: $E \rightarrow A_i$ repeated for all t attribute sorts A_is! **Examples:** Typical attributes of a person are Gender, Weight, Height, Birth date, etcetera. Dynamic and static attributes of a pipe unit include current flow into the unit, per input, if any, current flow out of the unit, per output, if any current leak from the unit, guaranteed maximum flow into the unit, guaranteed maximum flow out of the unit, guaranteed maximum leak from the unit, etcetera. Michael A. Jackson [49] categorizes attributes as either *static* or *dynamic*, with dynamic attributes being either *inert*, *reactive* or *active*. The latter are then either *autonomous*, *biddable* or *programmable.* This categorization has a strong bearing on how these (f.ex., part) attributes are dealt with when now interpreting parts as behaviours.

2.2 From Manifest Parts to Domain Behaviours

[35] then presents an interpretation, τ, which to manifest *parts* associate *behaviours*. These are then specified as CSP [48] *processes*. This interpretation amounts to a *transcendental deduction !*

The Transcendental Deduction Idea — by means of an example: The term *train* can have the following "meanings": The *train*, as an *endurant*, parked at the railway station platform, i.e., as a *composite part.* The *train*, as a *perdurant*, as it "speeds" down the railway track, i.e., as a *behaviour.* The *train*, as an *attribute*,

Atomic Parts: Atomic parts translate into their core behaviours: $b_{core}^{p_{atom}}$. The *core* behaviours are tail recursively defined, that is, are cyclic. $b_{core}^{p_{atom}}(...) \equiv (.... ; b_{core}^{p_{atom}}(...))$ where (...) indicate behaviour (i.e., function) arguments.

Composite Parts: A composite part, p, "translates", τ, into the parallel composition of a core behaviour: $b_{core}^{p_{comp}}(...)$, for part p, with the parallel composition of the translations, τ, for each of the parts, $p_1, p_2, ..., p_m$, of p, $(\tau(p_1)\|\tau(p_2)\|...\|\tau(p_m))$ that is: $\tau(p) \equiv b_{core}^{p_{comp}}(...)\|(\tau(p_1)\|\tau(p_2)\|...\|\tau(p_m))$

Concrete Parts: The translation of concrete part set, t, types, $t : T = K - \mathbf{set}$, is $\tau(t) \equiv \|\{\tau(k_i)|k_i{:}K \bullet k_i \in t\}$.

Translation of Part Qualities (...): Part qualities, that is: *unique identifiers*, *mereologies* and *attributes*, are translated into behaviour arguments – of one kind or another, i.e., (...). Typically we can choose to *index* behaviour names, b by the *unique identifier*, id, of the part based on which they were translated, i.e., b_{id}. *Mereology values* are usually static, and can, as thus, be treated like we treat static attributes (see next), or can be set by their behaviour, and are then treated like we treat programmable attributes (see next), i.e., (...). *Static attributes* become behaviour definition (body) constant values. *Inert, reactive* and *autonomous attributes* become references to channels, say *ch_dyn*, such that when an inert, reactive and autonomous attribute value is required it is expressed as *ch_dyn ?*. *Programmable* and *biddable attributes* become arguments which are passed on to the tail-recursive invocations of the behaviour, and possibly updated as specified [with]in the body of the definition of the behaviour, i.e., (...).

2.3 Contributions of [29] – and Open Problems

For the first time we have, now, the beginnings of a calculus for developing domain descriptions. In [13, 15] we speculate on laws that these analysis & description prompts (i.e., their "meanings") must satisfy. With this calculus we can now systematically develop domain descriptions [41–56]. I am right now working on understanding issues of *implicit*/*explicit* semantics[6] Since December 2017 I have revised [35] extensively: simplified it, extended it, clarified some issues, provided analysis & description techniques for channels and arguments, et cetera. The revised paper is [29][7].

[6] Cf. http://impex2017.loria.fr/

[7] You can find it on the Internet: http://www.imm.dtu.dk/~dibj/2018/tosem/Bjorner-TOSEM.pdf.

3 Related Papers

3.1 Domain Facets: Analysis & Description [31, 12]

Overview By a domain facet we shall understand one amongst a finite set of generic ways of analyzing 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.

[31] is an extensive revision of [12]. Both papers identify the following facets: *intrinsics*, *support technologies*, *rules & regulations*, *scripts*, *license languages*, *management & organisation*, and *human behaviour*. Recently I have "discovered" what might be classified as a domain facet: classes of *attribute semantics:* the diversity of attribute semantics resolving the issue of so-called implicit and explicit semantics. I shall not cover this issue in this talk.

Intrinsics: *By domain intrinsics we shall understand those phenomena and concepts of a domain which are basic to any of the other facets, with such domain intrinsics initially covering at least one specific, hence named, stakeholder view.*

Support Technology: *By a domain support technology we shall understand ways and means of implementing certain observed phenomena or certain conceived concepts.*

Rules and 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.

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 possibly has legally binding power, that is, which may be contested in a court of law.*

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) 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 "overseeable" 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.

Human Behaviour: *By domain human behaviour we shall understand any of a quality spectrum of carrying out assigned work: (i) from careful, diligent and accurate, via (ii) sloppy dispatch, and (iii) delinquent work, (iv) to outright criminal pursuit.*

Contributions of [31, 12] – and Open Problems: [31] now covers techniques and tools for analyzing domains into these facets and for their modeling. The issue of *license languages* are particularly intriguing. The delineations between the listed[8] facets is necessarily not as precise as one would wish: we are dealing with an imprecise world, that of (manifest) domains. License languages are treated in [31].

3.2 From Domains to Requirements [25, 8]

Overview: [25] outlines a calculus of refinements and extensions which applied to domain descriptions yield requirements prescriptions. As for [35] the calculus is to be deployed by human users, i.e., requirements engineers. Requirements are for a *machine*, that is, the hardware and software to be developed from the requirements. A distinction is made between *domain*, *interface* and *machine requirements*. I shall briefly cover these in another order.

[8] We have omitted a facet: *license languages*.

Machine requirements: Machine requirements are such which can be expressed using only technical terms of the machine: performance and dependability accessibility, availability, integrity, reliability, safety, security and robustness). and development requirements development process, maintenance, platform, management and documentation). Within *maintenance requirements* there are adaptive, corrective, perfective, preventive, and extensional requirements. Within *platform requirements* there are development, execution, maintenance, and demonstration requirements. Etcetera. [25] does not cover these. See instead [6, Sect. 19.6].

Domain Requirements: Domain requirements are such which can be expressed using only technical terms of the domain. The are the following domain-to-requirements specification transformations: *projection*, *instantiation*, *determination*, *extension* and *fitting*. *I consider my work on these domain requirements issues the most interesting.*

1: Projection: By a *domain projection* we mean *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.*

2: 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 the endurants: parts, materials and components, as well as the perdurants: actions, events and behaviours of the domain requirements prescription more concrete, more specific.*

3: 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 the endurants: parts, materials and components, as well as the perdurants: functions, events and behaviours of the partial domain requirements prescription less non-determinate, more determinate.*

4: Extension: By *domain extension* we understand *the introduction of endurants and perdurants 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.*

5: 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 harmonize the two or more software development projects by harmonizing, if not too late, their requirements developments.

Interface Requirements: *Interface requirements* are such which can be expressed only by using technical terms of both the domain and the machine. Thus interface requirements are about that which is *shared* between the domain and the machine: *endurants* that are represented in machine storage as well as co-existing in the domain; *actions* and *behaviours* that are performed while interacting with phenomena in the domain; etc.

Contributions of [25, 8]: [25] does not follow the "standard division" of requirements engineering into systems and user requirements etcetera. Instead [25] builds on domain descriptions and eventually gives a rather different "division of requirements engineering labour" – manifested in the domain, the interface and the machine requirements paradigms, and these further into sub-paradigms, to wit: projection, instantiation, determination, extension and fitting. Some readers have objected to my use of the term *refinement* for the domain-to-requirements transformations.

3.3 Formal Models of Processes and Prompts [23, 20]

Overview: [35] outlines a calculus of prompts, to be deployed by human users, i.e., the domain analyzers & describers. That calculus builds on the *assumption* that the domain engineers build, *in their mind*, i.e., *conceptually*, a *syntactical* structure of the domain description, although, what the domain engineers can "see & touch" are *semantic* objects. A formal model of the analysis and description prompt process and of the meanings of the prompts therefore is split into a model for the process and a model of the syntactic and semantics structures.

A Summary of Analysis and Description Prompts

The Analysis Prompts :

[a] is_entity	[f] is_part	[k] has_concrete_type
[b] is_endurant	[g] is_component	[l] has_mereology
[c] is_perdurant	[h] is_material	[m] has_components
[d] is_discrete	[i] is_atomic	[n] has_material
[e] is_continuous	[j] is_composite	[o] has_parts

The Description Prompts :

[1] observe_part_sorts	[5] observe_attributes
[2] observe_concrete_type	[6] observe_component_sorts
[3] observe_unique_identifier	[7] observe_part_material_sort
[4] observe_mereology	[8] observe_material_part_sorts

A Glimpse of the Process Model

Process "Management": Domain description involves the "generation" and use of an indefinite number of type (sort) names, Nm. The global, assignable variables αps and νps serve to hold the names of the sorts to be analysed, respectively the names of the sorts for which unique identifiers, mereologies and attributes have to be analysed and described.

type
    Nm = PNm | MNm | KNm
variable
    αps := [Δnm] **type** Nm-**set**
    νps := [Δnm] **type** Nm-**set**
value
    sel_and_remove_Nm: **Unit** → Nm
    sel_and_remove_Nm() ≡
        let nm:Nm • nm ∈ νps **in**
        νps := νps \ {nm} ; nm **end**; **pre**: νps ≠ {}

Some Process Functions: The analyse_and_describe_endurants function is the major function. It invokes a number of other analysis & description functions. We illustrate two:

value
  analyse_and_describe_endurants: **Unit** → **Unit**
  analyse_and_describe_endurants() ≡
    while ∼is_empty(νps) **do**
      let nm = sel_and_remove_Nm() **in**
      analyse_and_describe_endurant_sort(nm,ι:nm) **end end** ;
    for all nm:PNm • nm ∈ αps **do if** *has_mereology(nm,ι:nm)*
      then *observe_mereology*(nm,ι:nm) **end end**

for all nm:Nm • nm ∈ αps **do** *observe_attributes*(nm,ι:nm) **end**

analyse_and_describe_endurant_sort: NmVAL → **Unit**
analyse_and_describe_endurant_sort(nm,val) ≡
  is_part(nm,val) → *analyse_and_describe_part_sorts*(nm,val),
  is_material(nm,val) → *observe_material_part_sort*(nm,val),
  is_component(nm,val) → *observe_component_sort*(nm,val)

A Glimpse of the Syntax and Semantics Models We suggest a syntax and a semantics of domain descriptions.

The Syntactical Structure of Domains: First the syntax of domains – divided into the syntax of endurants parts, materials and components.

TypDef = PTypes ∪ MTypes ∪ KTypes
PTypes = PNm $\overrightarrow{m}$ PaTyp
MTypes = MNm $\overrightarrow{m}$ MaTyp
KTypes = KNm $\overrightarrow{m}$ KoTyp
ENDType = PaTyp | MaTyp | KoTyp
PaTyp == AtPaTyp | AbsCoPaTyp | ConCoPaTyp
AtPaTyp :: mkAtPaTyp(s_qs:PQ,s_omkn:({|"nil"|}|MNn|KNm))
AbsCoPaTyp :: mkAbsCoPaTyp(s_qs:PQ,s_pns:PNm-**set**)
axiom ∀ mkAbsCoPaTyp(pq,pns):AbsCoPaTyp • pns ≠ {}
ConCoPaTyp :: mkConCoPaTyp(s_qs:PQ,s_p:PNm)
MaTyp :: mkMaTyp(s_qs:MQ,s_opn:({|"nil"|}|PNm))
KoTyp :: mkKoTyp(s_qs:KQ)

Then the syntax of the internal qualities of endurants:

PQ = s_ui:UI×s_me:ME×s_atrs:ATRS}
UI
ME == "nil"|mkUI(s_ui:UI)|mkUIset(s_uil:UI)|...
ATRS = ANm $\overrightarrow{m}$ ATyp
ANm, ATyp
MQ = s_atrs:ATRS
KQ = s_uid:UI × s_atrs:ATRS

The Semantical Values of Domains: Corresponding, homomorphically, to these syntaxes are their semantics types:

$$
\begin{array}{rcl}
\mathsf{ENDVAL} & = & \mathrm{PVAL} \mid \mathrm{MVAL} \mid \mathrm{KVAL} \\
\mathsf{PVAL} & == & \mathrm{AtPaVAL}|\mathrm{AbsCoPVAL}|\mathrm{ConCoPVAL} \\
\mathsf{AtPaVAL} & :: & \mathrm{mkAtPaVAL(s\_qval:PQVAL,} \\
 & & \mathrm{s\_omkvals:(\{|"nil"|\}|MVAL|KVAL\text{-}\mathbf{set}))} \\
\mathsf{AbsCoPVAL} & :: & \mathrm{mkAbsCoPaVAL(s\_qval:PQVAL,s\_pvals:(PNm \underset{m}{\rightarrow} PVAL))} \\
\mathbf{axiom} & \forall & \mathrm{mkAbsCoPaVAL(pqs,ppm):AbsCoPVAL\bullet ppm \neq [\,]} \\
\mathsf{ConCoPVAL} & :: & \mathrm{mkConCoPaVAL(s\_qval:PQVAL,s\_pvals:PVAL\text{-}\mathbf{set})} \\
\mathsf{MVAL} & :: & \mathrm{mkMaVAL(s\_qval:MQVAL,s\_pvals:PVAL\text{-}\mathbf{set})} \\
\mathsf{KVAL} & :: & \mathrm{mkKoVAL(s\_qval:KQVAL)}
\end{array}
$$

Qualities: Semantic Types

$$
\begin{array}{rcl}
\mathsf{PQVAL} & = & \mathsf{UIVAL}\times\mathsf{MEVAL}\times\mathsf{ATTRVALS} \\
\mathsf{UIVAL} & & \\
\mathsf{MEVAL} & == & \mathsf{mkUIVAL(s\_ui:UIVAL)|mkUIVALset(s\_uis:UIVAL\text{-}\mathbf{set})|...} \\
\mathsf{ATTRVALS} & = & \mathsf{ANm} \underset{m}{\rightarrow} \mathsf{AVAL} \\
\mathsf{ANm,\ AVAL} & & \\
\mathsf{MQVAL} & = & \mathsf{ATTRVALS} \\
\mathsf{KQVAL} & = & \mathsf{UIVAL}\times\mathsf{ATTRVALS}
\end{array}
$$

From Syntax to Semantics and "Back Again!" We define mappings from sort names to the possibly infinite set of values of the named type, and from endurant values to the names of their sort.

$$
\begin{array}{l}
\mathbf{type} \\
\quad \mathsf{Nm\_to\_ENDVALS} = \\
\qquad \mathsf{(PNm \underset{m}{\rightarrow} PVAL\text{-}\mathbf{set})\cup(MNm \underset{m}{\rightarrow} MVAL\text{-}\mathbf{set})\cup(KNm \underset{m}{\rightarrow} KVAL\text{-}\mathbf{set})} \\
\quad \mathsf{ENDVAL\_to\_Nm} = \\
\qquad \mathsf{(PVAL \underset{m}{\rightarrow} PNm\ )\cup(MVAL \underset{m}{\rightarrow} MNm)\cup(KVAL \underset{m}{\rightarrow} KNm)} \\
\mathbf{value} \\
\quad \mathsf{typval:\ TypDef \stackrel{\sim}{\rightarrow} Nm\_to\_ENDVALS} \\
\quad \mathsf{typval(td)} \equiv \mathbf{let}\ \rho = \\
\qquad [\,\mathsf{n}\mapsto\mathsf{M(td(n))}(\rho)|\mathsf{n:(PNm|MNm|KNm)}\bullet\mathsf{n} \in \mathbf{dom}\ \mathsf{td}\,]\ \mathbf{in}\ \rho\ \mathbf{end} \\
\\
\quad \mathsf{valtyp:\ Nm\_to\_ENDVALS \stackrel{\sim}{\rightarrow} ENDVAL\_to\_Nm} \\
\quad \mathsf{valtyp}(\rho) \equiv \\
\qquad [\,\mathsf{v}\mapsto\mathsf{n|n:(PNm|MNm|CNm),v:(PVAL|MVAL|KVAL)}\bullet \\
\qquad\quad \mathsf{n} \in \mathbf{dom}\ \rho\wedge\mathsf{v} \in \rho(\mathsf{n})\,] \\
\\
\quad \mathsf{M:\ (PaTyp\rightarrow ENV\stackrel{\sim}{\rightarrow}PVAL\text{-}\mathbf{set})|} \\
\qquad \mathsf{(MaTyp\rightarrow ENV\stackrel{\sim}{\rightarrow}MVAL\text{-}\mathbf{set})|} \\
\qquad \mathsf{(KoTyp\rightarrow ENV\stackrel{\sim}{\rightarrow}KVAL\text{-}\mathbf{set})}
\end{array}
$$

The environment, ρ, of typval is the least fix point of the recursive equation. The crucial function is M, in the definition of typval. Examples of its definition, by part category, is given below.

value
  ι nm:Nm ≡ iota(nm)
  iota: Nm → TypDef → VAL
  iota(nm)(td) ≡
    let val:(PVAL|MVAL|KVAL)•val∈(typval(td))(nm)
    in val **end**

Analysis Functions: We exemplify the semantics functions for three analysis prompts.

value
  is_endurant: Nm×VAL → TypDef $\tilde{\to}$ **Bool**
  is_endurant(_,val)(td) ≡ val ∈ **dom** valtyp(typval(td));
      pre: VAL is any value type

  is_discrete: NmVAL → TypDef $\tilde{\to}$ **Bool**
  is_discrete(_,val)(td) ≡ (is_PaTyp|is_CoTyp)(td((valtyp(typval(td)))(val)))

  is_part: NmVAL → TypDef $\tilde{\to}$ **Bool**
  is_part(_,val)(td) ≡ is_PaTyp(td((valtyp(typval(td)))(val)))

Description Functions: We exemplify the semantics of one of the description prompts. The generated description **RSL-text** is enclosed within [" ... "].

variable
    τ := [ ] **Text-set**
value
    observe_part_sorts: Nm×VAL → TypDef → **Unit**
    observe_part_sorts(nm,val)(td) ≡
      let mkAbsCoPaTyp(_,{P_1,P_2,...,P_n})
        = td((valtyp(typval(td)))(val)) **in**
      $\tau := \tau \oplus$ [" **type** P_1,P_2,...,P_n;
            value
              obs_part_P_1: $nm \to P_1$
              obs_part_P_2: $nm \to P_2$
              ...,

obs_part_P_n: $nm \to P_n$;
proof obligation
$\mathcal{D}$; "]
$\parallel \nu ps := \nu ps \oplus$ ([" $P_1, P_2, ..., P_n$ "] $\setminus \alpha ps$)
$\parallel \alpha ps := \alpha ps \oplus$ [" $P_1, P_2, ..., P_n$ "]
end
pre: is_AbsCoPaTyp(td((valtyp(typval(td)))(val)))

.

The M Function

1 The meaning of an atomic part type expression,
   - mkAtPaTyp((ui,me,attrs),omkn) in
   - mkAtPaTyp(s_qs:PQ,s_omkn:({|"nil"|}|MNn|KNm)),
   - is the set of all atomic part values,
     mkAtPaVAL((uiv,mev,attrvals),omkval) in
   - mkAtPaVAL(s_qval:(UIVAL×MEVAL×(ANm $\overrightarrow{m}$ AVAL)),
     s_omkvals:({|"nil"|}|MVAL|KVAL-**set**)).

   a uiv is a value in UIVAL of type ui,
   b mev is a value in MEVAL of type me,
   c attrvals is a value in (ANm $\overrightarrow{m}$ AVAL) of type (ANm $\overrightarrow{m}$ ATyp), and
   d omkvals is a value in ({|"nil"|}|MVAL|KVAL-**set**):
      i either `''nil''`,
      ii or one material value of type MNm,
      iii or a possibly empty set of component values, each of type KNm.

```
1.    M: mkAtPaTyp((UI×ME×(ANm →m ATyp))×({|"nil"|}|MVAL|KVAL-set))
1.          →ENV ~→ PVAL-set
1.    M(mkAtPaTyp((ui,me,attrs),omkn))(ρ) ≡
1.       { mkATPaVAL((uiv,mev,attrval),omkvals) |
1a.         uiv:UIVAL•type_of(uiv)=ui,
1b.         mev:MEVAL•type_of(mev)=me,
1c.         attrval:(ANm →m AVAL)•type_of(attrval)=attrs,
1d.         omkvals: case omkn of
1(d)i.         "nil" → "nil",
1(d)ii.        mkMNn(_) → mval:MVAL•type_of(mval)=omkn,
1(d)iii.       mkKNm(_) →
1(d)iii.          kvals:KVAL-set•kvals⊆{kv|kv:KVAL•type_of(kval)=omkn}
1d.         end }
```

Formula terms 1a–1(d)iii express that any applicable uiv is combined with any applicable mev is combined with any applicable attrval is combined with any applicable omkvals.

2 The meaning of an abstract composite part type expression,
  - mkAbsCoPaTyp((ui,me,attrs),pns) in
  - mkAbsCoPaTyp(s_qs:PQ,s_pns:PNm-**set**), is the set of all abstract, composite part values,
  - mkAbsCoPaVAL((uiv,mev,attrvals),pvals) in
  - mkAbsCoPaVAL(s_qval:(UIVAL×MEVAL×(ANm $\overrightarrow{m}$ AVAL)), s_pvals:(PNm $\overrightarrow{m}$ PVAL)).

  a uiv is a value in UIVAL of type ui: UI,
  b mev is a value in MEVAL of type me: ME,
  c attrvals is a value in (ANm $\overrightarrow{m}$ AVAL) of type (ANm $\overrightarrow{m}$ ATyp), and
  d pvals is a map of part values in (PNm $\overrightarrow{m}$ PVAL), one for each name, pn:PNm, in pns such that these part values are of the type defined for pn.

```
2.  M: mkAbsCoPaTyp((UI×ME×(ANm →m ATyp)),PNm-set)
2.                  → ENV →~ PVAL-set
2.  M(mkAbsCoPaTyp((ui,me,attrs),pns))(ρ) ≡
2.     { mkAbsCoPaVAL((uiv,mev,attrvals),pvals) |
2a.        uiv:UIVAL•type_of(uiv)=ui
2b.        mev:MEVAL•type_of(mev)=me,
2c.        attrvals:(ANm →m ATyp)•type_of(attrsval)=attrs,
2d.        pvals:(PNm →m PVAL) •
2d.           pvals∈{[pn↦pval|pn:PNm,pval:PVAL•pn∈ pns∧pval∈ρ(pn)]} }
```

Contributions of [23] The contributions of [23] are to suggest and carry through a "formalisation" of the *conceptual, syntactical* and *semantical* structures *perceived* by the domain engineer, to formalise the *meaning of the informal* analysis & description prompts, and to formalise the possible sets of sequences of valid prompts.

3.4 To Every Manifest Domain Mereology a CSP Expression [33]

Overview In [35] we have shown how parts can be endowed with mereologies. Mereology, as was mentioned earlier, is the study and knowledge of *"part-hood":* of how parts are related parts to parts, and parts to *"a whole"*. Mereology, as treated by us, originated with the Polish mathematician/logician/philosopher Stanislaw Lešhniewski.

An Axiom System for Mereology :

$$\begin{aligned}
\textsf{part_of:} &\quad \mathbb{P} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}\\
\textsf{proper_part_of:} &\quad \mathbb{PP} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}\\
\textsf{overlap:} &\quad \mathbb{O} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}\\
\textsf{underlap:} &\quad \mathbb{U} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}\\
\textsf{over_crossing:} &\quad \mathbb{OX} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}\\
\textsf{under_crossing:} &\quad \mathbb{UX} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}\\
\textsf{proper_overlap:} &\quad \mathbb{PO} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}\\
\textsf{proper_underlap:} &\quad \mathbb{PU} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}
\end{aligned}$$

Let $\mathbb{P}$ denote ***part-hood***; p_x is part of p_y, is then expressed as $\mathbb{P}(p_x, p_y)$.[9] (1) Part p_x is part of itself (reflexivity). (2) 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). (3) 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).

$$\forall p_x : \mathcal{P} \bullet \mathbb{P}(p_x, p_x) \tag{1}$$

$$\forall p_x, p_y : \mathcal{P} \bullet (\mathbb{P}(p_x, p_y) \wedge \mathbb{P}(p_y, p_x)) \Rightarrow p_x = p_y \tag{2}$$

$$\forall p_x, p_y, p_z : \mathcal{P} \bullet (\mathbb{P}(p_x, p_y) \wedge \mathbb{P}(p_y, p_z)) \Rightarrow \mathbb{P}(p_z, p_z) \tag{3}$$

We exemplify one of the mereology propositions: ***proper underlap***, $\mathbb{PU}$: p_x and p_y are said to properly underlap if p_x and p_y under-cross and p_y and p_x under-cross.

$$\mathbb{PU}(p_x, p_y) \stackrel{\triangle}{=} \mathbb{UX}(p_x, p_y) \wedge \mathbb{UX}(p_y, p_x) \tag{4}$$

A Model for the Axioms [33] now gives a model for parts: atomic and composite, commensurate with [35] and [23], and their unique identifiers, mereology and attributes and show that the model satisfies the axioms.

Contributions of [33] [33] thus contributes to a domain science, helping to secure a firm foundation for domain engineering.

4 Domain Science & Engineering: A Philosophy Basis [30]

My most recent work is documented in [30]. It examines the question:

– *What must inescapably be in any domain description ?*

Another formulation is:

[9] Our notation now is not `RSL` but a conventional first-order predicate logic notation.

– *Which are the necessary characteristics of each and every possible world and our situation in it.*

Recent works by the Danish philosopher Kai Sørlander [52–55] appears to direct us towards an answer.

Here is how it is done, in brief. On the basis of *possibility of truth*[10] Sørlander establishes the logical connectors and from them the existence of a world with symmetry, asymmetry and transitivity. By a transcendental deduction Sørlander then reasons that *space* and *time*, inescapably, are "in the world"[11]. Further logical reasoning and transcendental deductions establishes the inescapability of *Newton's 1st, 2nd and 3rd Laws*. And from that *kinematics*, *dynamics*, and *gravitational pull*. And so forth. Thus the worlds that can possibly be described must all satisfy the *laws of physics*.

This line of reasoning and deduction thus justifies the focus, in our calculi, on natural parts, components and materials.

But Sørlander goes on and reasons and transcendentally deduce the inescapable existence of *living species: plants* and *animals*, and, among the latter, *humans*. Because of reasoned characteristics of humans we inescapably have *artifacts: man-made parts components* and *materials*. Humans construct artifacts with an *intent*, an attribute of both humans and artifacts. These *shared intents* lead to a notion of *intentional "pull"*[12] and so forth.

This line of reasoning and deduction thus justifies the inclusion, in our calculi, of living species and artifacts.

[30] is presently an approximately 90 page report. As such it is presently a repository for a number of "texts" related to the issue of *"what must inescapably be in any domain description ?"* It may be expected that a far shorter paper may emerge.

[10] Sørlander makes his *logical reasoning* and *transcendental deductions* on the basis of the *possibility of truth* – where Immanuel Kant [45], according to Sørlander, builds on the *possibility of self-awareness*, which is shown to lead to contradictions.

[11] Kant assumes space and time.

[12] We shall here give an example of *intentional "pull":* humans create automobiles and roads. An intention of automobiles is to drive on roads, and an intention of roads is to have automobiles move along roads. We can thus speak of the *traffic history of an automobile* as the time-stamped sequence of vehicle positions along roads, and of the *traffic history of a road* as the time-stamped sequence of vehicle positions along that road. Now, for the sum total of all automobiles and all roads the two consolidate histories must be identical. *It cannot be otherwise.*

5 The Experiments [41–56]

In order to test and tune the domain analysis & description method a great number of experiments were carried out. In our opinion, when applied to manifest domains, they justify the calculi reported in [35] and [23].

- *Urban Planning* [41],
- *A Space of Swarms of Drones* [27],
- *Documents* [28],
- *Credit Cards* [22],
- *Weather Information Systems* [26],
- *The Tokyo Stock Exchange* [34],
- *Pipelines* [18],
- *Road Transportation* [19],
- *Web/Transaction*-based Software [14],
- *"The Market"* [4],
- *Container [Shipping] Lines* [7],
- *Railway Systems* [3, 37, 5, 51, 56].

6 Summary

We have identified a discipline of domain science and engineering. Its first "rendition" was applied to the semantics of programming languages and the development of their compilers [46, CHILL] and [42, Ada]. Domain science and engineering, as outlined here, is directed at a wider spectrum of "languages": the "meaning" of computer application domains and software for these applications. Where physicists model facets of the world emphasizing physical, dynamic phenomena in nature, primarily using differential calculi, domain scientists cum engineers emphasize logical and both discrete phenomena of man and human institutions primarily using discrete mathematics.

7 Acknowledgments

I am grateful to Prof. Zhan NaiJun for inviting me to the Zhou ChaoChen Fest and for inviting me to submit my talk as a paper for this Festschrift. I am grateful to my "old student", now Prof. Ji Wang for his arranging a wonderful stay in Changsha, my fourth visit to that great city, and for his fantastic cheerful welcome.

8 Laudatio

At the Zhou ChaoChen Fest dinner I gave a dinner speech. It is not about Zhou Chaochen's scientific life. But it is a laudatio expressed in admiration for a wonderful man and our lives together.

- It was in 1981, in Beijing, 36 years ago. At the Institute of Computing Technology.
  On my first day of a three week visit. 30 lectures, 30 degrees Celsius. I liked it.
  I was being received. All sat in soft cushioned armchairs along the walls.
  I sat to the right of this wonderful man, Xu KongShi.
  During our conversation I queried about a young researcher, Zhou ChaoChen.
  Tony Hoare had told me to watch out for him.
  So I did, with an invitation letter, right in my pocket, for him to visit my dept.
  Asked Xu KongShi as to the whereabouts of Zhou?
  And he smiled: right next to the right of you !
- That became the first day of a 36 year acquaintance. Almost half of our life-times!
  Zhou came to visit us, 3 months every other winter. It was during the 1980s.
  What a wonderful time, for me, for my colleagues and for our students.
  One time I asked him to tutor a young MSc student. She performed brilliantly.
  It was something about "the meeting calendar problem". Even Zhou was impressed.
  Perhaps he has forgotten it now. When I took him to the airport, some weeks later.
  I told him that Ulla, that was her name, was a great granddaughter of Niels Bohr.
  Zhou appreciated then that I only told him then.
- For the 1989 visit I had "stipulated" that Zhou bring his family.
  Three months to Lyngby, three months to Oxford. And Zhou kindly agreed. All was set to go.
  But a certain incident early that June caused us all concern.
  Yet, on July 1st that year the whole family arrived.
- Zhou wasn't keen to return to China.
  I speed-dialled Tony's Oxford number.
  "Tony on the line" was the reply
  "Tony: Zhou is with me here, in my office in Lyngby."
  "Hello Zhou"
  "Hello Tony".
  "Tony, I have just offered Zhou a three year appointment."
  Well I hadn't, but there it was, and Zhou got listening.
  "8 months a year here at Lyngby. 4 with you at Oxford."
  Tony's reply: "Well, I had got it wrong, the other way around".
  "Let Zhou decide", I replied, and Zhou said: "It is as Dines proposes."
- Those became three great years, at Lyngby and at Oxford.
  Zhang Yi Ping and children lodged in Oxford - Zhou commuting. Science progressing.
  It was at a ProCoS meeting in Viborg.
  E.V.Sørensen had given a talk on signal transitions of electric circuits.
  The concept of 'duration' was mentioned.
  Afterwards I saw Zhou, A.P. and Tony, in an adjacent room.
  Discussing, standing at the white board, scribbling.
  And "The Duration Calculus" was born.
- The following year I was asked to become Director of UNU-IIST.
  On the flight home, in May 1991, from Japan, via a visit to Macau
  I decided to ask Zhou to join me in Macau.
  And a year later, things take time in international affairs, we began.
  With Zhou in charge of theory and I of engineering, an institute was built.
  After my five years followed Zhou's five years.
  Some of you, in this room, can look back at defining years at UNU-IIST.
  I returned to Lyngby and eventually Zhou to Beijing.

- The Duration Calculus took root.
  Painstakingly a theory was cemented and applications realized.
  The ProCoS project and UNU-IIST played an important rôle in this.
  But at the core of all this was Zhou ChaoChen.
- Dear Zhou:
  Thank you for your tremendous contributions to science.
  Thank you for inspiring generations of scientists.
  Thank you for hosting our daughter, Charlotte, the fall of 1986 – 31 years ago!
  Thank you for putting our son, Nikolaj, on the road to science – also 31 years ago!

9 Bibliography

9.1 Bibliographical Notes

In the last ten years I have also worked on related topics:

- Domains: Their Simulation, Monitoring and Control, see [16], [24] 2008,
- Compositionality: Ontology and Mereology of Domains[13], [36] 2008,
- Domain Science & Engineering, [13, 15] 2010,
- Computation for Humanity: Domain Science and Engineering, [17] 2012,
- 40 Years of Formal Methods — Obstacles and Possibilities[14], [39] 2014,
- Domain Engineering – A Basis for Safety Critical Software, [21] 2014,
- Implicit and Explicit Semantics and the Domain Calculi, [32] 2017.

Work on these papers and on the many, extensive experiments has helped solidify the basic domain analysis & description method.

9.2 References

1. Anon: C.C.I.T.T. High Level Language (CHILL), Recommendation Z.200, Red Book Fascicle VI.12. See [47], ITU (Intl. Telecmm. Union), Geneva, Switzerland (1980 – 1985)
2. Bjørner, D.: Programming Languages: Formal Development of Interpreters and Compilers. In: International Computing Symposium 77 (eds. E. Morlet and D. Ribbens). pp. 1–21. European ACM, North-Holland Publ.Co., Amsterdam (1977)
3. Bjørner, D.: Formal Software Techniques in Railway Systems. In: Schnieder, E. (ed.) 9th IFAC Symposium on Control in Transportation Systems. pp. 1–12. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik, Technical University, Braunschweig, Germany (13–15 June 2000), invited talk
4. Bjørner, D.: Domain Models of "The Market" — in Preparation for E–Transaction Systems. In: Practical Foundations of Business and System Specifications (Eds.: Haim Kilov and Ken Baclawski). Kluwer Academic Press, The Netherlands (December 2002), Final draft version. http://www2.imm.dtu.dk/ db/themarket.pdf

[13] with Asger Eir
[14] with Klaus Havelund

5. Bjørner, D.: Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In: CTS2003: 10th IFAC Symposium on Control in Transportation Systems. Elsevier Science Ltd., Oxford, UK (August 4-6 2003). https://doi.org/10.1016/S1474-6670(17)32424-2, symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki. Final version. http://www2.imm.dtu.dk/ db/ifac-dynamics.pdf
6. Bjørner, D.: Software Engineering, Vol. 3: Domains, Requirements and Software Design. Texts in Theoretical Computer Science, the EATCS Series, Springer (2006), See [9, 11]
7. Bjørner, D.: A Container Line Industry Domain. Techn. report, Fredsvej 11, DK-2840 Holte, Denmark (June 2007), Extensive Draft. http://www2.imm.dtu.dk/ db/container-paper.pdf
8. Bjørner, D.: From Domains to Requirements. In: Montanari Festschrift. Lecture Notes in Computer Science (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer), vol. 5065, pp. 1–30. Springer, Heidelberg (May 2008)
9. Bjørner, D.: Software Engineering, Vol. 3: Domains, Requirements and Software Design. Qinghua University Press (2008)
10. Bjørner, D.: On Mereologies in Computing Science. In: Festschrift: Reflections on the Work of C.A.R. Hoare. pp. 47–70. History of Computing (eds. Cliff B. Jones, A.W. Roscoe and Kenneth R. Wood), Springer, London, UK (2009)
11. Bjørner, D.: **Chinese:** Software Engineering, Vol. 3: Domains, Requirements and Software Design. Qinghua University Press. Translated by Dr Liu Bo Chao et al. (2010)
12. Bjørner, D.: Domain Engineering. In: Boca, P., Bowen, J. (eds.) Formal Methods: State of the Art and New Directions. pp. 1–42. Eds. Paul Boca and Jonathan Bowen, Springer, London, UK (2010). https://doi.org/10.1007/978-1-84882-736-3_1
13. Bjørner, D.: Domain Science & Engineering – *From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part*. Kibernetika i sistemny analiz (4), 100–116 (May 2010)
14. Bjørner, D.: On Development of Web-based Software: A Divertimento of Ideas and Suggestions. Technical, Technical University of Vienna (August–October 2010), http://www.imm.dtu.dk/˜dibj/wfdftp.pdf
15. Bjørner, D.: Domain Science & Engineering – *From Computer Science to The Sciences of Informatics Part II of II: The Science Part*. Kibernetika i sistemny analiz (2), 100–120 (May 2011)
16. Bjørner, D.: Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In: Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary., pp. 167–183. Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), Springer, Heidelberg, Germany (January 2011)
17. Bjørner, D.: Domain Science and Engineering as a Foundation for Computation for Humanity, chap. 7, pp. 159–177. Computational Analysis, Synthesis, and Design of Dynamic Systems, CRC [Francis & Taylor] (2013), (eds.: Justyna Zander and Pieter J. Mosterman)
18. Bjørner, D.: Pipelines – a Domain Description. http://www.imm.dtu.dk/˜dibj/pipe-p.pdf. Experimental Research Report 2013-2, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark (Spring 2013)
19. Bjørner, D.: Road Transportation – a Domain Description. http://www.imm.dtu.dk/˜dibj/road-p.pdf. Experimental Research Report 2013-4, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark (Spring 2013)
20. Bjørner, D.: Domain Analysis: Endurants – An Analysis & Description Process Model. In: Iida, S., Meseguer, J., Ogata, K. (eds.) Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi. Springer (May 2014)

21. Bjørner, D.: Domain Engineering – A Basis for Safety Critical Software. Invited Keynote, ASSC2014: Australian System Safety Conference, Melbourne, 26–28 May (December 2014)
22. Bjørner, D.: A Credit Card System: Uppsala Draft. Technical Report: Experimental Research, Fredsvej 11, DK–2840 Holte, Denmark (November 2016), http://www.imm.dtu.dk/˜dibj/2016/credit/accs.pdf
23. Bjørner, D.: Domain Analysis and Description – Formal Models of Processes and Prompts (2016), extensive revision of [20]. http://www.imm.dtu.dk/˜dibj/2016/-process/process-p.pdf
24. Bjørner, D.: Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. Tech. rep., Fredsvej 11, DK–2840 Holte, Denmark (2016), extensive revision of [16]. http://www.imm.dtu.dk/˜dibj/2016/demos/faoc-demo.pdf
25. Bjørner, D.: From Domain Descriptions to Requirements Prescriptions – A Different Approach to Requirements Engineering (2016), Extensive revision of [8]
26. Bjørner, D.: Weather Information Systems: Towards a Domain Description. Technical Report: Experimental Research, Fredsvej 11, DK–2840 Holte, Denmark (November 2016), http://www.imm.dtu.dk/˜dibj/2016/wis/wis-p.pdf
27. Bjørner, D.: A Space of Swarms of Drones. Research Note (November–December 2017), http://www.imm.dtu.dk/˜dibj/2017/docs/docs.pdf
28. Bjørner, D.: What are Documents? Research Note (July 2017), http://www.imm.-dtu.dk/˜dibj/2017/docs/docs.pdf
29. Bjørner, D.: A Domain Analysis & Description Method – Principles, Techniques and Modeling Languages. Research Note based on [35] (February 20 2018), http://www.imm.dtu.dk/ dibj/2018/tosem/Bjorner-TOSEM.pdf
30. Bjørner, D.: A Philosophy of Domain Science & Engineering – An Interpretation of Kai Sørlander's Philosophy. Research Note (Spring 2018), http://www.imm.dtu.dk/˜dibj/-2018/philosophy/filo.pdf
31. Bjørner, D.: Domain Facets: Analysis & Description (May 2018), extensive revision of [12]. http://www.imm.dtu.dk/˜dibj/2016/facets/faoc-facets.pdf
32. Bjørner, D.: The Manifest Domain Analysis & Description Approach to Implicit and Explicit Semantics. EPTCS: Electronic Proceedings in Theoretical Computer Science, Yasmine Ait-Majeur, Paul J. Gibson and Dominique Méry (2018), First International Workshop on Handling IMPlicit and EXplicit Knowledge in Formal Fystem Development, 17 November 2017, Xi'an, China
33. Bjørner, D.: To Every Manifest Domain a CSP Expression — A Rôle for Mereology in Computer Science. Journal of Logical and Algebraic Methods in Programming (94), 91–108 (January 2018). https://doi.org/10.1016/j.jlamp.2017.09.005
34. Bjørner, D.: The Tokyo Stock Exchange Trading Rules. R&D Experiment, Fredsvej 11, DK-2840 Holte, Denmark (January and February, 2010), Version 1. http://www2.imm.dtu.dk/ db/todai/tse-1.pdf, Version 2. http://www2.imm.dtu.dk/ db/todai/tse-2.pdf
35. Bjørner, D.: Manifest Domains: Analysis & Description. Formal Aspects of Computing **29**(2), 175–225 (Online: July 2016). https://doi.org/10.1007/s00165-016-0385-z
36. Bjørner, D., Eir, A.: Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In: Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness. Lecture Notes in Computer Science, vol. 5930, pp. 22–59. Springer, Heidelberg (July 2010)
37. Bjørner, D., George, C.W., Prehn, S.: Computing Systems for Railways — A Rôle for Domain Engineering. Relations to Requirements Engineering and Software for

Control Applications. In: Integrated Design and Process Technology. Editors: Bernd Kraemer and John C. Petterson. Society for Design and Process Science, P.O.Box 1299, Grand View, Texas 76050-1299, USA (24–28 June 2002), Extended version. http://www2.imm.dtu.dk/ db/pasadena-25.pdf
38. Bjørner, D., Gram, C., Oest, O.N., Rystrømb, L.: Dansk Datamatik Center. In: Wangler, B., Lundin, P. (eds.) History of Nordic Computing. Springer, Stockholm, Sweden (18-20 October 2010)
39. Bjørner, D., Havelund, K.: 40 Years of Formal Methods — 10 Obstacles and 3 Possibilities. In: FM 2014, Singapore, May 14-16, 2014. Springer (2014), Distinguished Lecture
40. Bjørner, D., Oest, O.N. (eds.): Towards a Formal Description of Ada, LNCS, vol. 98. Springer (1980)
41. Bjørner, D.: Urban Planning Processes. Research Note (July 2017), http://www.imm.-dtu.dk/~dibj/2017/up/urban-planning.pdf
42. Clemmensen, G., Oest, O.: Formal specification and development of an Ada compiler – a VDM case study. In: Proc. 7th International Conf. on Software Engineering, 26.-29. March 1984, Orlando, Florida. pp. 430–440. IEEE (1984)
43. George, C.W., Haff, P., Havelund, K., Haxthausen, A.E., Milne, R., Nielsen, C.B., Prehn, S., Wagner, K.R.: The RAISE Specification Language. The BCS Practitioner Series, Prentice-Hall, Hemel Hampstead, England (1992)
44. George, C.W., Haxthausen, A.E., Hughes, S., Milne, R., Prehn, S., Pedersen, J.S.: The RAISE Development Method. The BCS Practitioner Series, Prentice-Hall, Hemel Hampstead, England (1995)
45. Guyer, P. (ed.): The Cambridge Companion to Kant. Cambridge Univ. Press, England (1992)
46. Haff, P., Olsen, A.: Use of VDM within CCITT. In: VDM – A Formal Method at Work, eds. Dines Bjørner, Cliff B. Jones, Micheal Mac an Airchinnigh and Erich J. Neuhold, pp. 324–330. Springer, Lecture Notes in Computer Science, Vol. 252 (March 1987), Proc. VDM-Europe Symposium 1987, Brussels, Belgium
47. Haff, P. (ed.): The Formal Definition of CHILL. ITU (Intl. Telecmm. Union), Geneva, Switzerland (1981)
48. Hoare, C.: Communicating Sequential Processes. C.A.R. Hoare Series in Computer Science, Prentice-Hall International (1985), published electronically: http://www.usingcsp.com/cspbook.pdf (2004)
49. Jackson, M.A.: Software Requirements & Specifications: a lexicon of practice, principles and prejudices. ACM Press, Addison-Wesley, Reading, England (1995)
50. Oest, O.N.: VDM from research to practice (invited paper). In: IFIP Congress. pp. 527–534 (1986)
51. Pěnička, M., Strupchanska, A.K., Bjørner, D.: Train Maintenance Routing. In: FORMS'2003: Symposium on Formal Methods for Railway Operation and Control Systems. L'Harmattan Hongrie (15–16 May 2003), conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. Final version. http://www2.imm.dtu.dk/ db/martin.pdf
52. Sørlander, K.: Det Uomgængelige – Filosofiske Deduktioner [The Inevitable – Philosophical Deductions, with a foreword by Georg Henrik von Wright]. Munksgaard · Rosinante (1994), 168 pages
53. Sørlander, K.: Under Evighedens Synsvinkel [Under the viewpoint of eternity]. Munksgaard · Rosinante (1997), 200 pages
54. Sørlander, K.: Den Endegyldige Sandhed [The Final Truth]. Rosinante (2002), 187 pages

55. Sørlander, K.: Indføring i Filosofien [Introduction to The Philosophy]. Informations Forlag (2016), 233 pages
56. Strupchanska, A.K., Pěnička, M., Bjørner, D.: Railway Staff Rostering. In: FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems. L'Harmattan Hongrie (15–16 May 2003), conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany. Final version. http://www2.imm.dtu.dk/ db/albena.pdf