Domain Engineering
“Upstream” from Requirements Engineering and Software Design

Foil # 1

Dines Bjørner
Department of Computer Science & Technology
Technical University of Denmark
DK–2800 Lyngby, Denmark
E–Mail: db@it.dtu.dk

June 2, 2000

Abstract

Before software can be developed its requirements must be stated. Before requirements can be expressed the application domain must be described. In this document we outline some of the basic facets of domain engineering.

Domains seem, it is our experience, far more stable than computing requirements, and these again seem more stable then software designs. Perhaps a way in which to more rapidly develop trustworthy software from believable requirements is to secure comprehensive domain theories. Foil # 2

An brief example will be given on the basis of which we briefly discuss, in this document, the notions of: domain intrinsics, domain support technologies, domain management & organisation, domain rules & regulations, domain human behaviour, etc. We show elsewhere how to “derive” requirements from domain descriptions: domain requirements: by domain projection, instantiation, extension and initialisation; interface requirements: multi–media, dialogue, etc.; and machine requirements: performance, dependability (reliability, availability, accessability, safety, etc.). The current document presents *work-in-progress.*

Contents

1 Background

Foil # 3

The workshop for which this modest contribution is drawn up takes it departure point in the US President's Information Technology Advisory Committee (PITAC) 1998 interim report: *Need for software outstrip development resources. Desparately needed software is not being developed. Software must be made far more usable, reliable and powerful. Current development, test and maintenance processes must change. Scientifically sound software development approaches are required: Enabling meaningful and practical testing for consistency of specificsations and implementations.* The current document, based on [1, 2, 3, 4, 5, 6, 7, 8, 9] immodestly suggest that a stronger emphasis need be put, in future, on domain engineering as one means of reaching the PITAC goals.

2 Example: Resource Management

Foil # 4

2.1 Synopsis and Narrative

The scope is that of resources and their management. The span is that of strategic, tactical and operations management and of actual operations. Strategic resource management is about acquiring ("expansion, upgrading") or disposing ("down-sizing, divestiture") of resources: Converting one form of resource to another. Tactical resource management is about allocating resources spatially and scheduling them for general, temporal availability. Operations resource management is about allocating resources to tasks and scheduling them for special, time interval deployment. These three kinds of resource management reflect rather different perspectives: Strategic resource management is the prerogative and responsability of executive management. Tactical resource management is the prerogative and responsability of line ("middle level") management. Operations resource management is the prerogative and responsability of operations (ie. "ground level") management.

2.2 Formalisation: Resources and their Handling

Foil # 5

2.2.1 Formalisation: Resources

type R, Rn, L, T, E, A
  RS = R-**set**
  SR = T $\overrightarrow{m}$ RS,   SRS = SR-**infset**
  TR = (T×T) $\overrightarrow{m}$ R $\overrightarrow{m}$ L,   TRS = TR-**set**
  OR = (T×T) $\overrightarrow{m}$ R $\overrightarrow{m}$ A

  A = (Rn $\overrightarrow{m}$ R-**set**) $\tilde{\to}$ (Rn $\overrightarrow{m}$ R-**set**)

value
  srm: RS → E×E $\tilde{\to}$ E × (SRS × SR)
  trm: SR → E×E $\tilde{\to}$ E × (TRS × TR)
  orm: TR → E×E $\tilde{\to}$ E × OR

  ope: OR → TR → SR → (E×E×E×E) → E × RS

  p: E → **Bool**

srm, trm and orm are the strategic, tactical and operations management functions. ope is the actual operations function. p is a predicate which determines whether the enterprise can continue to operate (eith its state and in its environment, e, or not.

2.2.2 Resource Formalisation — Annotation

Foil # 6

R, L, T, E and A stand for resources, spatial locations, times, the enterprise (with its estimates, service and/or production plans, orders on hand, etc.), respectively tasks (actions). SR, TR and OR stand for strategic, tactical and operational resource views, respectively. srm, trm and orm stand for strategic, tactical, respectively operations resource management. To keep our model "small", we have had to resort to a "trick": Putting all the facts knowable and needed in order for management to function adequately into E ! E, besides the enterprise itself, also models its environment: That part of the world which affects the enterprise. *Foil # 7*

There are, accordingly, the following management functions: *Strategic resource management,* srm(rs)(e,e'''')=(e',(srs,sr)), proceed on the basis of the enterprise (e) and its current resources (rs), and "ideally estimates" all possible strategic resource possibilities (srs), and selects one, desirable (sr). The "estimation" is heuristic. Too little is normally known to compute sr algorithmically. We refer to [5] for details. *Tactical resource management,* trm(sr)(e,e'''')=(e'',(trs,tr)), proceed on the basis of the enterprise (e) and one chosen strategic resource view (sr) and "ideally calculates" all possible tactical resource possibilities (trs), and selects one, desirable (tr). As for strategic resource management, we refer to [5] for details.

Operations resource management, orm(tr)(e,e'''')=(e''',or), proceed on the basis of the enterprise (e) and one chosen tactical resource view (tr) and effectively decides on one operations resource view (or). We refer to [5] for details. *Actual enterprise operation*, ope, enables, but does not guarantee, some "common" view of the enterprise: ope depends on the views of the enterprise its state and environment, as "passed down" by management; and ope applies, according to presecriptions kept in the enterprise state, actions, a, to named (rn:Rn) sets of resources.

2.2.3 Formalisation: Resource Handling

Foil # 8

The above account is, obviously, rather "idealised". But, hopefully, indicative of what is going on. To give a further abstraction of the "life cycle" of the enterprise we "idealise" it as now shown:

```
value
  enterprise: RS -~-> E -~-> Unit
  enterprise(rs)(e) ==
    if p(e) then
      let (e',(srs,sr)) = srm(rs)(e,e''''),
          (e'',(trs,tr)) = trm(sr)(e,e''''),
          (e''',or) = orm(tr)(e,e''''),
          (e'''',rs') = ope(or)(tr)(sr)(e,e',e'',e''') in
      let e''''':E • p'(,e'''',e''''') in
      enterprise(rs')(e''''') end end
    else stop end

  p': E × E → Bool
```

The enterprise re-invocation argument, rs′, a result of operations, is intended to reflect the use of strategially, tatctically and operationally acquired, spatially and task allocated and scheduled resources, including partial consumption, "wear & tear", loss, replacements, etc.

Foil # 9

An imperative version of enterprise could be:

```
value
  enterprise: E → RS → Unit
  enterprise(e)(rs) ==
    variable ve:E := e;
    while p(ve) do
      let (e',(srs,sr)) = srm(rs)(ve,e''''),
          (e'',(trs,tr)) = trm(sr)ve,(e''''),
          (e''',or) = orm(tr)(ve,e''''),
          (e'''',rs') = ope(or)(tr)(sr)(ve,e',e'',e''') in
      let e''''':E • p'(e'''',e''''') in
      ve := e''''' end end end

  ope: OR → TR → SR → (E×E×E×E) → E × RS
```

Only the program flow of control recursion has been eliminated. The **let** $e''''':\mathsf{E} \bullet \mathsf{p}'(e'''', e''''')$ **in** ... shall model a changing environment.

2.2.4 Resource Handling — Annotation

Foil # 10

There are two forms of recursion at play here: The simple tail-recursive, next step, day-to--day recursion, and the recursive "build-up" of the enterprise state e''''. The latter is the interesting one. To solve it, by iteration towards some acceptable, not necessarily minimal fixpoint, the three levels of management and the "floor" operations change that state and "pass it around, up-&-down" the management "hierarchy". The operate function "unifies" the views that different management levels have of the enterprise, and influences their decision making. Dependence on E also models potential interaction betwen enterprise management and, conceivably, all other stake-holders. We remind the reader that we are "only" modelling the domain — with all its imperfections !

3 Discussion

Foil # 11

The model just presented is, obviously, sketchy. But we believe it portrays important facets of domain modelling.

We are modelling a domain with all its imperfections: We are not specifying anything algorithmically; all functions are rather loosely deined, in fact only their signature is given. This means that we model well-managed as well as badly, sloppily, disastrously managed enterprises. We can, of course, define a great number of predicates on the enterprise state and its environment (e:E), and we can partially characterise intrinsics — facts that must always be true of en enterprise, no matter how well or badly it is managed. But if we "programme--specified" the enterprise then we would not be modelling the domain of enterprises, but a specifically "business process engineered" enterprise. And we would be into requirements engineering — we claim. So let us take now, a closer view of the kind of things we can indeed model in the domain !

4 Stake–holder Perspectives

Foil # 12

There are several kind of domain stake-holders: Enterprise stake–holders: (i) owners, (ii) management: (a) executive, (b) line, and (c) "floor", managers, (iii) workers, (iv) families of the above. Non–enterprise stake–holders: (v) clients (customers), (vi) competitors, resource providers: (a) IT resource providers, (b) non–IT/non–finance resource providers, and (c) financial service providers, (vii) regulatory agencies, (viii) politicians, and (ix) the "public–at–large". For each stake-holder there usually is a distinct domain perspective: A partial specification. The example shown earlier illustrated, at some level of abstraction, the interaction between, hence the perspectives of, enterprise managers and workers, and, at a higher level of abstraction, interaction with the environment, incl. all other stake-holders. *Foil # 13*

5 Domain Facets

Foil # 14

We shall sketch the following facets:

Domain intrinsics: That which is common to all facets.

Domain support technologies: That in terms of which most other facets (intrinsics, management, organisation, and rules & regulations) are implemented.

Domain management and organisation: That which constrains communication between enterprise stake-holders.

Domain rules & regulations: That which guides the work of enterprise stake-holders as well as their interaction and the interaction with non-enterprise stake-holders.

Domain human behaviour: The way in which domain stake-holders despatch their actions and interactions wrt. enterprise: dutifully, forgetfully, sloppily, yes even criminally.

We shall briefly characterise each of these facets. We venture to express a "specification pattern" that "most closely captures" an essence of the facet.

5.1 Intrinsics

Foil # 15

The intrinsics of a rail switch is that it can take on a number of states. A simple switch $({}^{c_|}Y_c^{c_/})$ has three connectors: $\{c, c_|, c_/\}$. c is the connector of the common rail from which one can either "go straight" $c_|$, or "fork" $c_/$.

$$\begin{array}{lll} \omega & : & \{\ \ \{\}, \\ & & \{(c,c_|)\},\{(c,c_|),(c_|,c)\},\{(c_|,c)\}, \\ & & \{(c,c_/)\},\{(c,c_/),(c_/,c)\},\{(c_/,c)\}, \\ & & \{(c,c_/),(c_|,c)\},\{(c,c_/),(c_/,c),(c_|,c)\},\{(c_/,c),(c_|,c)\}\ \ \} \end{array}$$

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 signals up or down the rail away from the switch.

Foil # 16

The intrinsics of a domain is a partial specification. Amongst many specification patterns the following is relevant.

type
  Γ_i, Σ_i, Syntax, VAL_i
value
  I: Syntax $\rightarrow$ Γ_i $\tilde{\rightarrow}$ Σ_i $\tilde{\rightarrow}$ Σ_i
  V: Syntax $\rightarrow$ Γ_i $\tilde{\rightarrow}$ Σ_i $\tilde{\rightarrow}$ VAL
  E: Syntax $\rightarrow$ Γ_i $\tilde{\rightarrow}$ Σ_i $\tilde{\rightarrow}$ Σ_i $\times$ VAL_i
  D: Syntax $\rightarrow$ Γ_i $\tilde{\rightarrow}$ Σ_i $\tilde{\rightarrow}$ Γ_i $\times$ Σ_i
  C: Syntax $\rightarrow$ Γ_i $\tilde{\rightarrow}$ Σ_i $\tilde{\rightarrow}$ Γ_i $\times$ Σ_i $\times$ VAL_i

Other specification patterns, for example centered around concurrent processes, hence having "result" type **Unit,** are also likely.

Intrinsics descriptions emphasise looseness and non-determinism.

5.2 Support Technologies

Foil # 17

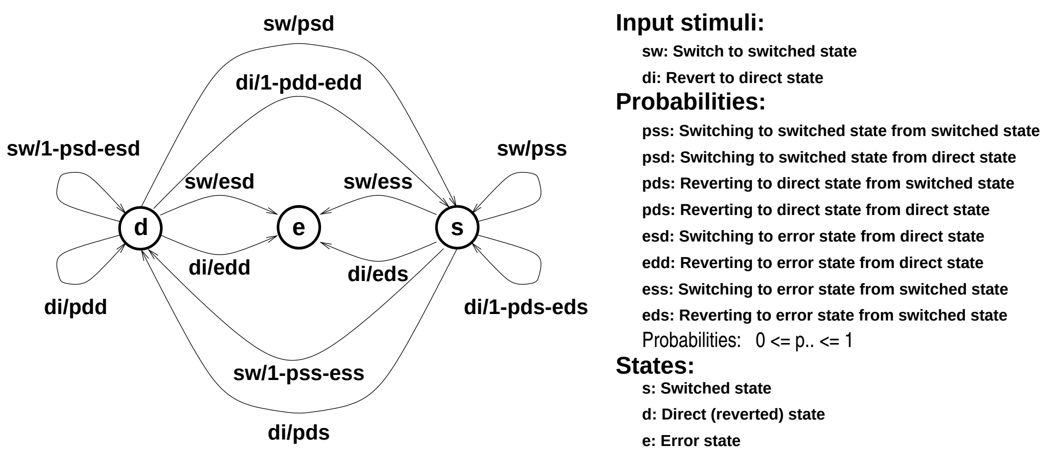
An example of different technology *stimuli:* A railway switch, "in ye olde days" of the "childhood" of railways, was *manually "thrown";* later it could be mechanically controlled from a distance by *wires and momentum "aplification";* Again later it could be electro-mechanically controlled from a further distance by *electric signals that then activated mechanical controls;* and today switches are usually *controlled in groups that are electronically interlocked.*

Foil # 18

Example: Probabilistic Rail Switch State Machine

An aspect of supporting technology includes the recording of state-behaviour in response to external stimuli. Figure 1 indicates a way of formalising this aspect of a supporting technology.

Figure 1: State Switching



Foil # 19

Example: Air Traffic Radar

Air traffic (iAT), intrinsically, is a total function over some time interval, from time (T) to monotonically positioned (P) aircrafts (A). A conventional air traffic controllers radar “samples”, at regular intervals, the intrinsic air traffic. Hence a radar is a partial function[1] from intrinsic to sampled air traffics (sAT).

type
  iAT = T → (A $\underset{m}{\to}$ P)
  sAT = T $\underset{m}{\to}$ (A $\underset{m}{\to}$ P)
value
  radar: iAT $\stackrel{\sim}{\to}$ sAT
  close: P × P → **Bool**
axiom
  ∀ iat:iAT •
    let sat = radar(iat) **in**
    ∀ t:T • t ∈ **dom** sat •

[1]This example is due to my current MSc Thesis student Kristian M. Kalsing

t ∈ **dom** iat ∧ ∀ a:A • a ∈ **dom** iat(t) ⇒
  a ∈ **dom** sat(t) ∧ close((iat(t))(a),(sat(t))(a)) **end**

An axiom relates intrinsic air traffic to radar sampled air traffic. The axiom thus characterises this support technology.

Foil # 20

Support technologies thus "implement" contexts and states: $\gamma_i : \Gamma_i, \sigma_i : \Sigma_i$ in terms of "actual" contexts and states: $\gamma_a : \Gamma_a, \sigma_a : \Sigma_a$

type
  Syntax,
  Γ_i, Σ_i, VAL_i,
  Γ_a, Σ_a, VAL_a,
  ST = Γ_i × Σ_i $\tilde{\to}$ Γ_a × Σ_a
value
  sts:ST-**set**
  I: Syntax → Γ_a $\tilde{\to}$ Σ $\tilde{\to}$ Σ_a
  V: Syntax → Γ_a $\tilde{\to}$ Σ $\tilde{\to}$ VAL_a
  E: Syntax → Γ_a $\tilde{\to}$ Σ $\tilde{\to}$ Σ_a × VAL_a
  D: Syntax → Γ_a $\tilde{\to}$ Σ $\tilde{\to}$ Γ_a × Σ_a
  C: Syntax → Γ_a $\tilde{\to}$ Σ $\tilde{\to}$ Γ_a × Σ_a × VAL_a

Support technology is not a refinement. Support technology typically introduces considerations of technology accuracy, failure, etc. Axioms characterise members of the set of support technologies sts.

5.3 Management and Organisation

Foil # 21

People staff the enterprises, the components of infrastructures with which we are concerned: For which we develop software. The larger these enterprises, these infrastructure components, are, the more need there is for management and organisation. The rôle of management is roughly, for our purposes, twofold: To make strategic, tactical and operational policies and see to it that they are followed, and to react to adverse conditions: Unforeseen situations, and decide upon their handling.

Foil # 22

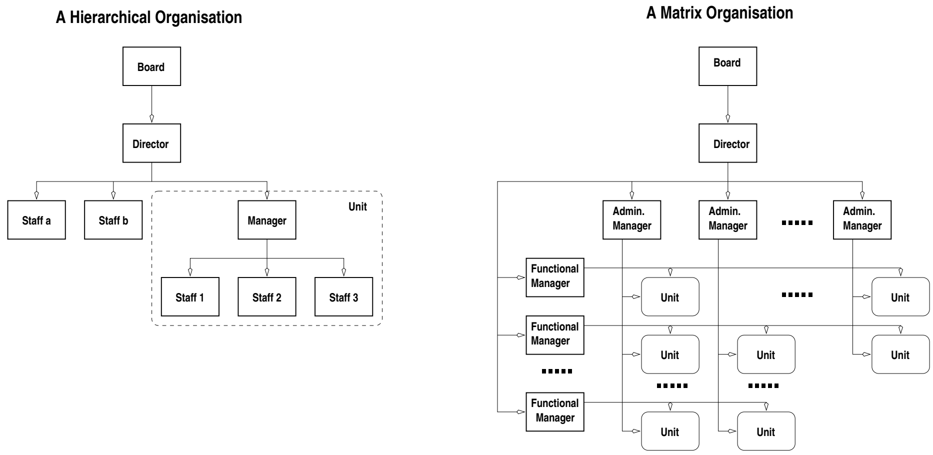
Policy setting should help non-management staff operate normal situations — for which no management interference is thus needed, and management "back-stops" problems: Takes these problems off the shoulders of non-management staff. To help management and staff know who's in charge wrt. policy setting 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 follow the command line: The paths of organigrams — the usually hierarchical box and arrow/line diagrams.

Foil # 23

Management is a set of predicates, observer and generator functions which parameterise other, the operations functions, that is: determine their behaviour. Organisation is a set of constraints on communication behaviours.

Foil # 24

Figure 2: Organisational Structures




```
type
   Msg, Ψ, Σ
channel
   { ms[i]:Msg | i:Sx }
value
   sys: Unit → Unit
   mgr: Ψ → in,out  { ms[i] | i:Sx } Unit
   stf: i:Sx → Σ → in,out ms[i] Unit

   sys() ≡ ‖ { stf(i)(iσ) | i:Sx } ‖ mgr(ψ)

   mgr(ψ) ≡
      let ψ′ = ...
           (‖ { ms[i]!msg | i:Sx } ...)
           ⌷
           (⌷ { let msg′ = ms[i]? in ... end | i:Sx })
             ... in mgr(ψ′) end

   stf(i)(σ) ≡
      let σ′ = ...
           (let msg = ms[i]? in f(msg)(σ) end)
           ⌷
           (... ms[i]!msg′ ... ) ... in stf(σ′) end
```

Foil # 25

The example of Section 2 illustrated another management & organisation description pattern. It is based on a set of, in this case, recursive equations. Any way of solving these equations, finding a suitable fixpoint, or an approximation thereof, including just choosing and imposing an arbitrary "solution", reflects some management communication. The syntactic ordering of the equations — in this case: a "linear" passing of enterprise "results" from "upper" equations onto "lower" equations — reflects some organisation. More "hierarchical", than "linear", or "matrix" structured organisations can also be modelled as sets (of recursively invoked sets) of equations.

We refrain, in this case from showing the two forms of specification patterns that here characterise management & organisation. The examples should suffice.

5.4 Rules & Regulations

Foil # 26

In China arrival and departure of trains at, respectively from railway stations are subject to the following regulation: In any three minute interval at most one train may either arrive or depart.

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 — obviously in the same direction — along the line, then there must be at least one free sector (ie. without a train) between any two such trains.

In the United State of America personal checks issued in any one state of the union must be cleared by the sending and receiving banks, if within the same state, then within 24 hours, and else within 48 or 72 hours, depending on certain further stipulated relations between the states.

Foil # 27

There may be two kinds of syntax involved here: The syntax (Syntax_rr) describing the rules & regulations. And the syntax (Syntax_cmd) of [always current] system "input". A rule or regulation is, semantically, a predicate over [current] system input (i:Syntax_cmd) and current and next-state system configuration. We omit treatment of [current] system "input", hence choose RR rather than RR′ below:

type
  Syntax_cmd, Syntax_rr
  $\mathrm{RR}' = \mathrm{Syntax\_cmd} \to (\Gamma\times\Gamma) \to (\Sigma\times\Sigma) \to \mathbf{Bool}$
  $\mathrm{RR} = (\Gamma\times\Gamma) \to (\Sigma\times\Sigma) \to \mathbf{Bool}$
  RRS = RR-**set**
value
  interpret: Syntax_rr → RRS

  valid: $\mathrm{RRS} \to (\Gamma\times\Gamma) \to (\Sigma\times\Sigma) \to \mathbf{Bool}$
  $\mathrm{valid(rrs)}(\gamma,\gamma')(\sigma,\sigma') \equiv \forall\ \mathrm{rr{:}RR} \bullet \mathrm{rr} \in\ \mathrm{rrs} \Rightarrow \mathrm{rr}(\gamma,\gamma')(\sigma,\sigma')$

5.5 Human Behaviour

Foil # 28

Some people try their best to perform actions according to expectations set by their colleagues, customers, etc. And they usually succeed in doing so. They are therefore judged reliable and trustworthy, good, punctual professionals (b_p) of their domain. Some people set lower standards for their professional performance: Are sometimes or often sloppy (b_s), make mistakes, unknowingly or even knowingly. And yet other people are outright delinquent (b_d) in the despatch of their work: Could'nt care less about living up to expectations of their colleagues and customers. Finally some people are explicitly criminal (*b_c*) in the conduct of what they do: Deliberately "do the opposite" of what is expected, circumvent rules & regulations, etc. And we must abstract and model, in any given situation where a human interferes in the "workings" of a domain action, any one of the above possible behaviours ! *Foil # 29*

We often model the "arbitrariness", the impredictability, of human behaviour by internal non-determinism:

$$\ldots \text{ b\_p} \sqcap \text{b\_s} \sqcap \text{b\_d} \sqcap \text{b\_c} \ldots$$

The exact, possibly deterministic, meaning of each of the b's can be separately described.

In addition we can model human behaviour by the arbitrary selection of elements from sets and subsets of sets:

let x:X • s ∈ xs **in** ... **end**
let xs′:X-**set** • xs′ ⊆ xs **in** ... **end**

The above shows just a fragment of a formal description of that part which reflects human behaviour. *Foil # 30*

Commensurate with the above, humans interpret rules & regulations differently, and not always "consistently" in the sense of repeatedly applying the same interpretations. Our final specification pattern is therefore:

type
  Action = Γ $\stackrel{\sim}{\to}$ Σ $\stackrel{\sim}{\to}$ Γ × Σ
value
  interpret: Syntax_rr → RRS-**infset**

  human_behaviour: Action → (Γ×Syntax_rr) $\stackrel{\sim}{\to}$ Σ $\stackrel{\sim}{\to}$ Γ × Σ
  human_behaviour(α)(γ,srr)(σ) **as** (γ′,σ′)
    post
      α(γ)(σ) = (γ′,σ′) ∧
      let rrs:RRS • rrs ∈ interpret(srr) **in**
        ∀ rr:RR • rr ∈ rrs ⇒ rr(γ,γ′)(σ,σ′) **end**

We have taken some liberties in expressing the action part, but the iedea should be clear: Humans determine next configurations based on arbitrary interpretation of rules & regulations.

5.6 Discussion

Foil # 31

We have sketched some domain facets: Intrinsics, support technologies, management and organisation, rules & regulations, and human behaviour. For each we attempted a characterising specification pattern. There are, undoubtedly other facets. Much work needs to be done.

Elsewhere ([10, 1]) we detail other facets: (I) Abstraction & modelling facets: (i) Model– and property-oriented specifications, (ii) representational and operational abstractions, (iii) functional, imperative, logic and concurrent models, (iv) hierachical and compositional developments and presentations, (iv) denotational and computational semantics, (v) configurations: context and state models, (vi) determinacy and looseness; (II) domain attributes: (vii) Discrete, continuous and chaotic domain, (viii) static and dynamic (inert, active [autonomous, biddable, programmable], and reactive) domain [11], and (ix) tangible and intangible domain [11]; and (III) domain frames (Jackson: [11], and [4].) (x) algebraic structures (languages), (xi) reactive systems, (xii) information systems ("databases"), (xiii) workflows (flexible manufacturing, healthcare, ofice automation, etc.), (xiv) workpieces (CAD, word processing, administration forms, etc.) (xv) connectors, *&c.* Whether these "axes of description" are comprehensive remains to be seen. *Foil # 32*

6 Conclusion

Foil # 33

We have presented work-in-progress.

The work reported here is part of an endeavour to understand *"What is a method ?".* Here we define a method to be a set of principles for *selecting* and *applying* a number of techniques and tools in order `efficiently` to *analyse* and *synthesize* and `efficient` 'artefact' — here: software. One can formalise certain technques and one can base certain tools on such formal techniques. Among the most common application of formal techniques and tools in software development is specification and analysis (incl. verification). Methods cannot be formal: Human enginuity, which principles are chosen in relation to others, cannot be formalised.

We endeavour to have shown you some principles, some techniques and a very few tools (RSL and finite state, probabilistic automata).

6.1 Acknowledgements

Foil # 34

I am grateful for the inspiration drawn from the work of Michael Jackson on requirements and software specification. I am grateful to my other colleagues in IFIP WG 2.2 for like inspiration. And I am grateful to my current students: Asger Eir, Kristian M. Kalsing, et al.

6.2 Thanks

Thanks are due to US/ARO, US/NSF, US/ARO–Europe, and INDAM (GNIM) for their co-sponsorskip of this workshop. Thanks are due to Manfred Broy, LuQi, Carlo Ghezzi and Zohar Manna for co-chairing the PC. But very special and deeply warm and appreciative thanks are due to: Gianna Reggio and Egidio Astesiano of, and the University of Genoa for their indefatigable work, respectively support in bringing us all together.

7 Bibliographical Notes

This document being a very preliminary draft lacks proper citations.

I do, however, strongly confess my delight in having studied Jackson's [12, 11, 13, 14, 15].

To support the implied claims made in the present document we refer to the following own reports and publications: [1, 2, 3, 5, 6, 7, 8, 9, 10, 4].

References

[1] Dines Bjørner. Domain Engineering, Elements of a Software Engineering Methodology — Towards Principles, Techniques and Tools — A Study in Methodology. Research report, Dept. of Computer Science & Technology, Technical University of Denmark, Bldg. 343, DK–2800 Lyngby, Denmark, 2000. One in a series of summarising research reports [2, 3].

[2] Dines Bjørner. Requirements Engineering, Elements of a Software Engineering Methodology — Towards Principles, Techniques and Tools — A Study in Methodology. Research report, Dept. of Computer Science & Technology, Technical University of Denmark, Bldg. 343, DK–2800 Lyngby, Denmark, 2000. One in a series of summarising research reports [1, 3].

[3] Dines Bjørner. Software Design: Architectures and Program Organisation, Elements of a Software Engineering Methodology — Towards Principles, Techniques and Tools — A Study in Methodology. Research report, Dept. of Computer Science & Technology, Technical University of Denmark, Bldg. 343, DK–2800 Lyngby, Denmark, 2000. One in a series of summarising research reports [1, 2].

[4] Dines Bjørner, Souleymane Koussoube, Roger Noussi, and Gueorgui Satchok. Jackson's Problem Frames: Domain, Requirements and Design. In Shaoying Liu, editor, *International Conference on Formal Engineering Methods: ICFEM'97*, Washington D.C., USA, 12–14 November 1997. IEEE Computer Science Press; IEEE sponsored conference, Hiroshima, Japan.

[5] Dines Bjørner. Domain Modelling: Resource Management Strategics, Tactics & Operations, Decision Support and Algorithmic Software. In J.C.P. Woodcock, editor, *Festschrift to Tony Hoare*. Oxford University and Microsoft, September 13–14 1999.

[6] Dines Bjørner. Pinnacles of Software Engineering: 25 Years of Formal Methods. *Annals of Software Engineering*, 1999. Eds. Dilip Patel and Wang ....

[7] Dines Bjørner. A Triptych Software Development Paradigm: Domain, Requirements and Software. Towards a Model Development of A Decision Support System for Sustainable Development. In ErnstRüdiger Olderog, editor, *Festschrift to Hans Langmaack*. University of Kiel, Germany, October 1999.

[8] Dines Bjørner. Where do Software Architectures come from ? Systematic Development from Domains and Requirements. A Re–assessment of Software Engneering ? *South African Journal of Computer Science*, 1999. Editor: Chris Brink.

[9] Dines Bjørner and Jorge R. Cuéllar. Software Engineering Education: Rôles of formal specification and design calculi. *Annals of Software Engineering*, 6:365–410, 1998. Published April 1999.

[10] Dines Bjørner. Domains as Prerequisites for Requirements and Software *&c.* In M. Broy and B. Rumpe, editors, *RTSE'97: Requirements Targeted Software and Systems Engineering*, volume 1526 of *Lecture Notes in Computer Science*, pages 1–41. Springer-Verlag, Berlin Heidelberg, 1998.

[11] Michael Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley Publishing Company, Wokingham, nr. Reading, England; E-mail: ipc@awpub.add-wes.co.uk, 1995. ISBN 0-201-87712-0; xiv + 228 pages.

[12] Michael Jackson. Problems, methods and specialisation. *Software Engineering Journal*, pages 249–255, November 1994.

[13] M. Jackson. Problems and requirements (software development). In *Second IEEE International Symposium on Requirements Engineering (Cat. No.95TH8040)*, pages 2–8. IEEE Comput. Soc. Press, 1995. .

[14] Michael Jackson. The meaning of requirements. *Annals of Software Engineering*, 3:5–21, 1997.

[15] P. Zave and M. Jackson. Four dark corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology*, 6(1):1–30, January 1997. .