Domain Theory: Practice and Theories*
A Discussion of Possible Research Topics

Dines Bjørner†

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Technical University of Denmark
DK-2800 Kgs. Lyngby, Denmark‡

Abstract. By a domain we mean a universe of discourse.

Typical examples are (partially) man-made universes of discourse - such as Air Traffic, Airports, Financial Services (banks, insurance companies, securities trading [brokers, traders, stock exchanges]), Health Care (hospitals etc.), Secure IT Systems (according to Intl. ISO/IEC Standard 17799), The Market (consumers, retailers, wholesalers, producers, "the supply chain"), Transportation (road, air, sea and/or rail transport), etc.

We shall outline how one might describe such (infrastructure component) domains, informally and formally - what the current descriptional limitations appear to be, and, hence, the prospects for future research as well as practice.

The current paper is based on Part IV, Chaps. 8–16 of [3]. The volume is one of [1–3].

The aim of this paper is to suggest a number of areas of domain theory and methodology research.

Maybe the title of the paper need be explained: The second part of the title: 'Practice and Theories' shall indicate that there is an engineering practice (i.e., methodology) of developing domain descriptions and that any such domain description forms the basis for a specific domain theory. The first part of the title: 'Theories' shall indicate that we need support the practice, i.e., the methodology, by theoretical insight, and that there probably are some theoretical insight that applies across some or all domain theories.

1 Introduction

1.1 A Preamble

This paper is mostly a computing science paper. This paper is less of a computer science paper. Computer science is the study and knowledge about the "things"

*Invited paper for ICTAC 2007, The 4th International Colloquium on Theoretical Aspects of Computing, 26–28 September 2007, Macau SAR, China: http://www.iist.unu.edu/ictac07/.

†Prof. Emeritus

‡Home address: Fredsvej 11, DK-2840 Holte, Denmark

that can exist "inside" computers, and of what computing is. Computing science is the study and knowledge about how to construct computers and the "things" that can exist "inside" computers. Although the main emphasis of 'Domain Theory and Practice' is computing science, some of the research topics identified in this paper have a computer science nature.

1.2 On Originality

Since this paper is an invited paper and since it basically builds on and extends a certain part (Part IV Domain Engineering) of Vol. 3, [3], of my book [1–3], I shall not bring a lot of motivation nor putting my possible contributions in a broader context other that saying this: as far as I can see from the literature my concept of domain engineering is new. It may have appeared in rudimentary forms here and there in the literature, but in the nine chapters (Chaps. 8–16) of Part IV, [3], it receives a rather definitive and fully comprehensive treatment. But even that treatment can be improved. The present paper is one such attempt.

1.3 Structure of Paper

In a first semi-technical section we briefly express the triptych software engineering dogma, its consequences and its possibilities. We relate software verification to the triptych and present a first research topic. Then we list some briefly explained domains, and we present three more research topics. In the main technical section of this paper we present five sets of what we shall call domain facets (intrinsics, support technology, management and organisation, rules and regulations, and human behaviour). Each of these will be characterised but not really exemplified. We refer to [3] for details. But we will again list corresponding research topics. The paper ends first with some thoughts about what a 'domain theory' is, then on relations to requirements, and finally on two rather distinct benefits from domain engineering. In that final part of the paper we discuss a programming methodology notion of 'requirements specific development models' and its research topics.

2 Domain Engineering: A Dogma and its Consequences

2.1 The Dogma

First the dogma: Before software can be designed its requirements must be understood. Before requirements can be prescribed the application domain must be understood.

2.2 The Consequences

Then the "idealised" consequences: In software development we first describe the domain, then we prescribe the requirements, and finally we design the software.

As we shall see: major parts of requirements can be systematically "derived"[1] from domain descriptions. In engineering we can accommodate for less idealised consequences, but in science we need investigate the "ideals".

2.3 The Triptych Verification

A further consequence of this triptych development is that

$$\mathcal{D}, \mathcal{S} \models \mathcal{R},$$

which we read as: in order to prove that $\mathcal{S}$oftware implements the $\mathcal{R}$equirements the proof often has to make assumptions about the $\mathcal{D}$omain.

2.4 Full Scale Development: A First Suggested ℜesearch Topic

Again, presupposing much to come we can formulate a first research topic.

ℜ 1. **The $\mathcal{D}, \mathcal{S} \models \mathcal{R}$ Relation**: Assume that there is a formal description of the $\mathcal{D}$omain, a formal prescription of the $\mathcal{R}$equirements and a formal specification of the $\mathcal{S}$oftware design. Assume, possibly, that there is expressed and verified a number of relations between the $\mathcal{D}$omain description and the $\mathcal{R}$equirements prescription. Now how do we express the assertion: $\mathcal{D}, \mathcal{S} \models \mathcal{R}$ — namely that the software is correct? We may assume, without loss of generality, that this assertion is in some form of a pre/post condition of $\mathcal{S}$ — and that this pre/post condition is supported by a number of assertions "nicely spread" across the $\mathcal{S}$oftware design (i.e., the code). The research topic is now that of studying how, in the pre/post condition of $\mathcal{S}$ (the full code) and in the (likewise pre/post condition) assertions "within" $\mathcal{S}$, the various components of $\mathcal{R}$ and $\mathcal{D}$ "appear", and of how they relate to the full formal pre- and descriptions, respectively.

2.5 Examples of Domains

The Examples. Lest we loose contact with reality it is appropriate here, however briefly, to give some examples of (application) domains.

Air Traffic: A domain description includes descriptions of the entities, functions, events and behaviours of aircraft, airports (runways, taxi-ways, apron, etc.), air lanes, ground, terminal, regional, and continental control towers, of (national [CAA, CAAC, FAA, SLV, etc.] and international [JAA, CAO]) aviation authorities, etc.

Airports: A domain description includes descriptions of the flow of people (passengers, staff), material (catering, fuel, baggage), aircraft, information (boarding cards, baggage tags) and control; of these entities, of the operations

[1] By "derivation" we here mean one which is guided by humans (i.e., the domain and requirements engineers in collaboration with the stakeholders).

performed by or on them, the events that may occur (cancellation or delay of flights, lost luggage, missing passenger), and, hence, of the many concurrent and intertwined (mutually "synchronising") behaviours that entities undergo.

Container Shipping: A domain description includes descriptions of containers, container ships, the stowage of containers on ships and in container yards, container terminal (ports), the loading and unloading of containers between ships and ports and between ports and the "hinterland" (including crances, port trucking and feeder trucks, trains and barges), the container bills of lading (or way bills), the container transport logistics, the (planning and execution, scheduling and allocation) of voyages, the berthing (arrival and departure) of container ships, customer relations, etc.

Financial Service Industry: A domain description includes descriptions of banks (and banking: [demand/deposit, savings, mortgage] accounts, [opening, closing, deposit, withdrawal, transfer, statements] operations on accounts), insurance companies (claims processing, etc.), securities trading (stocks, bonds, brokers, traders, exchanges, etc.), portfolio management, IPOs, etc.

Health care: A domain description includes descriptions of the entities, operations, events and behaviours of healthy people, patients and medical staff, of private physicians, medical clinics, hospitals, pharmacies, health insurance, national boards of health, etc.

The Internet: The reader is encouraged to fill in some details here!

Manufacturing: Machining & Assembly: The reader is encouraged to also fill in some details here!

"The" Market: A domain description includes descriptions of the entities, operations, events and behaviours of consumers, retailers, wholesalers, producers, the delivery chain and the payment of (or for) merchandise and services.

Transportation: A domain description includes descriptions of the entities, functions, events and behaviours of transport vehicles (cars/trucks/busses, trains, aircraft, ships), [multimodal] transport nets (roads, rail lines, air lanes, shipping lanes) and hubs (road intersections [junctions], stations, airports, harbours), transported items (people and freight), and of logistics (scheduling and allocation of transport items to transport vehicles, and of transport vehicles to transport nets and hubs). Monomodal descriptions can focus on just air traffic or on container shipping, or on railways.

The Web: The reader is encouraged to "likewise" fill in some details here!

There are many "less grand" domains: railway level crossings, the interconnect cabling between the oftentimes dozens of "boxes" of some electronic/mechanical/acoustical measuring set-up, a gas burner, etc. These are all, rather one-sidedly, examples of what might be called embedded, or real-time, or safety critical systems.

We can refer to several projects at UNU-IIST which have produced domain specifications for railway systems (China), ministry of finance (Vietnam), telephone systems (The Philippines), harbours (India), etc.; and to dozens of MSc projects which have likewise produced domain specifications for airports, air traffic, container shipping, health care, the market, manufacturing, etc. I give many,

many references in [3]. I also refer the reader to http://www.railwaydomain.org/ for documents, specifically http://www.railwaydomain.org/book.pdf for domain models of railway systems.

Some Remarks. A point made by listing and explaining the above domains is the following: They all display a seeming complexity in terms of multitude of entities, functions, events and interrelated behaviours; and they all focus on the reality of "what is out there": no mention is (to be) made of requirements to supporting computing systems let alone of these (incl. software).

2.6 Domains: Suggested ℜesearch Topics

From the above list we observe that the 'transportation item' "lifts" those of 'air traffic' and 'container shipping'. Other examples could be shown. This brings us, at this early stage where we have yet to really outline what domain engineering is, to suggest the following research topics:

ℜ 2. **Lifted Domains and Projections**: We observe, above, that the 'transportation' domain seems to be an abstraction of at least four more concrete domains: road, rail, sea and air transportation. We could say that 'transportation' is a commensurate "lifting" of each of the others, or that these more concrete could arise as a result of a "projection" from the 'transportation' domain. The research topic is now to investigate two aspects: a computing science cum software engineering aspect and a computer science aspect. The former should preferably result in principles, techniques and tools for choosing levels of "lifted" abstraction and "projected" concretisation. The latter should study the implied "lifting" and "projection" operators.

ℜ 3. **What Do We Mean by an Infrastructure ?** We observe, above, that some of the domains exemplify what is normally called infrastructure[2] components. According to the World Bank: *'Infrastructure' is an umbrella term for many activities referred to as 'social overhead capital' by some development economists, and encompasses activities that share technical and economic features (such as economies of scale and spillovers from users to nonusers).* The research is now to study whether we can reformulate the sociologically vague World Bank definition in precise mathematical terms.

ℜ 4. **What Is an Infrastructure Component ?** We observe, above, that not all of the domains exemplified are what is normally called infrastructure

[2]Winston Churchill is quoted to have said, during a debate in the House of Commons, in 1946: *... The young Labourite speaker that we have just listened to, clearly wishes to impress upon his constituency the fact that he has gone to Eton and Oxford since he now uses such fashionable terms as 'infra-structures'. [I have recently been in communication with the British House of Commons information office enquiries manager, Mr. Martin Davies in order to verify and, possibly pinpoint, this statement. I am told that "as the Hansard debates in question are not available electronically, it could only be found via a manual search of hard copy Hansard". So there it stands.]*

components.[3] The research is now to study whether we can formulate and formalise some "tests" which help us determine whether some domain that we are about to model qualifies as part of one or more infrastructure components.

We bring these early research topic suggestions so that the reader can better judge whether domain engineering principles and techniques might help in establishing a base for such research. Throughout the paper we shall "spice it" with further suggestions of research topics.

• • •

We do not cover the important methodological aspects of stakeholder identification and liaison, domain acquisition and analysis, domain model verification and validation. For that we refer to Vol. 3 Chaps. 9–10 and 12–14 [3].

3 Domain Facets

The rôle, the purpose, of domain engineering is to construct, to develop, and research domain descriptions. It is both an engineering and a scientific task. It is engineering because we do know, today, a necessary number of principles, techniques and tools with which to create domain models. It is scientific, i.e., of research nature, because, it appears, that we do not necessarily know, today, whether what we know is sufficient.

3.1 Stages of Domain Development

By domain development we mean a process, consisting of a number of reasonably clearly separable stages which when properly conducted leads to a domain description, i.e., a domain model. We claim that the following are meaningful and necessary domain development stages of development, each with their attendant principles, techniques and tools: (i) identification of stakeholders, (ii) rough domain identification, (iii) domain acquisition, (iv) analysis of rough domain description units, (v) domain modelling, (vi) domain verification, (vii) domain validation and (viii) domain theory formation. We shall focus on domain modelling emphasising the modelling concept of domain facets.

3.2 The Facets

By domain modelling we mean the construction of both an informal, narrative and a formal domain description.

We claim that the following identified facets (i.e., "steps") (later to be briefly explained) are necessary parts of the domain modelling process: (i) intrinsics,

[3] 'Manufacturing' and 'The Market' appear, in the above list to not be infrastructure components, but, of course, they rely on the others, the infrastructure components.

(ii) support technologies, (iii) management and organisation, (iv) rules and regulations, (v) and human behaviour. Ideally speaking one may proceed with these "steps" in the order listed. Engineering accommodates for less ideal progressions. Each "step" produces a partial domain description. Subsequent "steps" 'extend' partial descriptions into partial or even (relative) complete descriptions.

In this section, Sect. 3, we will not give concrete examples but will rely on such already given in Chap. 11 of [3].

3.3 Intrinsics

By the intrinsics of a domain we shall understand those phenomena and concepts, that is, those entities, functions, events and behaviours in terms of which all other facets are described.

The choice as to what constitutes the intrinsics of a domain is often determined by the views of the stakeholders. Thus it is a pragmatic choice, and the choice cannot be formalised in the form of an **is_intrinsics** predicate that one applies to phenomena and concepts of the domain.

Intrinsics: Suggested ℜesearch Topic.

ℜ 5. **Intrinsics**: We refer to Sect. 11.3 in [3]. What is, perhaps, needed, is a theoretically founded characterisation of "being intrinsic".

3.4 Support Technology

By a support technology of a domain we shall understand either of a set of (one or more) alternative entities, functions, events and behaviours which "implement" an intrinsic phenomenon or concept. Thus for some one or more intrinsic phenomena or concepts there might be a technology which supports those phenomena or concepts.

Sampling Behaviour of Support Technologies. Let us consider **i**ntrinsic **A**ir **T**raffic as a continuous function (→) from **T**ime to **F**light **L**ocations:

type
T, F, L
iAT = T → (F $\overrightarrow{m}$ L)

But what is observed, by some support technology, is not a continuous function, but a discrete sampling (a map $\overrightarrow{m}$):

sAT = T $\overrightarrow{m}$ (F $\overrightarrow{m}$ L)

There is a support technology, say in the form of **radar** which "observes" the intrinsic traffic and delivers the sampled traffic:

value
radar: iAT → sAT

Probabilistic cum Statistical Behaviour of Support Technologies. But even the radar technology is not perfect. Its positioning of flights follows some probabilistic or statistical pattern:

```
type
  P = {|r:Real • 0≤r≤1|}
  ssAT = P m→ sAT-infset
value
  radar′: iAT ~→ ssAT
```

The radar technology will, with some probability produce either of a set of samplings, and with some other probability some other set of samplings, etc.[4]

Support Technology Quality Control, a Sketch. How can we express that a given technology delivers a reasonable support ? One approach is to postulate intrinsic and technology states (or observed behaviours), Θ_i, Θ_s, a support technology τ and a "**close**ness" predicate:

```
type
  Θ_i, Θ_s
value
  τ: Θ_i → P m→ Θ_s-infset
  close: Θ_i × Θ_s → Bool
```

and then require that an experiment can be performed which validates the support technology.

The experiment is expressed by the following axiom:

```
value
  p_threshhold:P
axiom
  ∀ θ_i:Θ_i •
    let pθ_ss = τ(θ_i) in
    ∀ p:P • p>p_threshhold ⇒
      θ_s:Θ_s • θ_s ∈ pθ_ss(p) ⇒ close(θ_i,θ_s) end
```

The p_threshhold probability has to be a-priori determined as one above which the support technology renditions of the intrinsic states (or behaviours) are acceptable.

Support Technologies: Suggested ℜesearch Topics.

[4] Throughout this paper we omit formulation of type well-formedness predicates.

ℜ 6. **Probabilistic and/or Statistical Support Technologies**: Some cases should be studied to illuminate the issue of probability versus statistics. More generally we need more studies of how support technologies "enter the picture", i.e., how "they take over" from other facet. And we need to come up with precise modelling concepts for probabilistic and statistical phenomena and their integration into the formal specification approaches at hand.

ℜ 7. **A Support Technology Quality Control Method**: The above sketched a 'support technology quality control' procedure. It left out the equally important 'monitoring' aspects. Develop experimentally two or three distinct models of domains involving distinct sets of support technologies. Then propose and study concrete implementations of 'support technology quality monitoring and control' procedures.

3.5 Management and Organisation

By the management of an enterprise (an institution) we shall understand a (possibly stratified, see 'organisation' next) set of enterprise staff (behaviours, processes) authorised to perform certain functions not allowed performed by other enterprise staff (behaviours, processes) and where such functions involve monitoring and controlling other enterprise staff (behaviours, processes). By organisation of an enterprise (an institution) we shall understand the stratification (partitioning) of enterprise staff (behaviours, processes) with each partition endowed with a set of authorised functions and with communication interfaces defined between partitions, i.e., between behaviours (processes).

An Abstraction of Management Functions. Let **E** designate some enterprise state concept, and let **stra_mgt, tact_mgt, oper_mgt, wrkr** and **merge** designate strategic management, tactical management, operational management and worker actions on states such that these actions are "somehow aware" of the state targets of respective management groups and or workers. Let **p** be a predicate which determines whether a given target state has been reached, and let **merge** harmonise different state targets into an agreeable one. Then the following behaviour reflects some aspects of management.

```
type
  E
value
  stra_mgt, tact_mgt, oper_mgt, wrkr, merge: E×E×E×E → E
  p: E* → Bool
  mgt: E → E
  mgt(e) ≡
    let e′ = stra_mgt(e,e″,e‴,e⁗),
        e″ = tact_mgt(e,e″,e‴,e⁗),
        e‴ = oper_mgt(e,e″,e‴,e⁗),
        e⁗ = wrkr(e,e″,e‴,e⁗) in
```

if p(e,e″,e‴,e⁗)
  then skip
  else mgt(merge(e,e″,e‴,e⁗))
end end

The recursive set of $e^{\prime\cdot\cdot\prime} = f(e, e'', e''', e'''')$ equations are "solved" by iterative communication between the management groups and the workers. The arrangement of these equations reflect the organisation and the various functions, **stra_mgt, tact_mgt, oper_mgt** and **wrkr** reflect the management. The frequency of communication between the management groups and the workers help determine a quality of the result.

The above is just a very crude, and only an illustrative model of management and organisation.

We could also have given a generic model, as the above, of management and organisation but now in terms of, say, CSP processes. Individual managers are processes and so are workers. The enterprise state, $e : E$, is maintained by one or more processes, separate from manager and worker processes. Etcetera.

Management and Organisation: Suggested ℜesearch Topics.

ℜ 8. **Strategic, Tactical and Operation Management**: We made no explicit references to such "business school of administration" "BA101" topics as 'strategic' and 'tactical' management. Study Example 9.2 of Sect. 9.3.1 of Vol. 3 [3]. Study other sources on 'Strategic and Tactical Management'. Question Example 9.2's attempt at delineating 'strategic' and 'tactical' management. Come up with better or other proposals, and/or attempt clear, but not necessarily computable predicates which (help) determine whether an operation (above they are alluded to as 'stra' and 'tact') is one of strategic or of tactical concern.

ℜ 9. **Modelling Management and Organisation: Applicatively or Concurrently**: The abstraction of 'management and organisation' on Page 3.5 was applicative, i.e., a recursive function — whose auxiliary functions were hopefully all continuous. Suggest a CSP rendition of "the same idea" ! Relate the applicative to the concurrent models.

3.6 Rules and Regulations

By a rule of an enterprise (an institution) we understand a syntactic piece of text whose meaning apply in any pair of actual present and potential next states of the enterprise, and then evaluates to either true or false: the rule has been obeyed, or the rule has been (or will be, or might be) broken. By a regulation of an enterprise (an institution) we understand a syntactic piece of text whose meaning, for example, apply in states of the enterprise where a rule has been broken, and when applied in such states will change the state, that is, "remedy" the "breaking of a rule".

Abstraction of Rules and Regulations. Stimuli are introduced in order to capture the possibility of rule-breaking next states.

type
  Sti, Rul, Reg
  RulReg = Rul × Reg
  Θ
  STI = $\Theta \to \Theta$
  RUL = $(\Theta \times \Theta) \to$ **Bool**
  REG = $\Theta \to \Theta$
value
  meaning: Sti → STI, Rul → RUL, Reg → REG
  valid: Sti × Rul → $\Theta \to$ **Bool**
  valid(sti,rul)$\theta \equiv$ (meaning(rul))(θ,meaning(sti)θ)
axiom
  ∀ sti:Sti,(rul,reg):RulReg,θ:Θ •
    ∼valid(sti,rul)$\theta \Rightarrow$ meaning(rul)(θ,meaning(reg)θ)

Quality Control of Rules and Regulations. The axiom above presents us with a guideline for checking the suitability of (pairs of) rules and regulations in the context of stimuli: for every proposed pair of rules and regulations and for every conceivable stimulus check whether the stimulus might cause a breaking of the rule and, if so, whether the regulation will restore the system to an acceptable state.

Rules and Regulations Suggested ℜesearch Topic:.

ℜ 10. **A Concrete Case**: The above sketched a quality control procedure for 'stimuli, rules and regulations'. It left out the equally important 'monitoring' aspects. Develop experimentally two or three distinct models of domains involving distinct sets of rules and regulations. Then propose and study concrete implementations of procedures for quality monitoring and control of 'stimuli, rules and regulations'.

3.7 Human Behaviour

By human behaviour we understand a "way" of representing entities, performing functions, causing or reacting to events or participating in behaviours. As such a human behaviour may be characterisable on a per phenomenon or concept basis as lying somewhere in the "continuous" spectrum from (i) diligent: precise representations, performances, event (re)actions, and behaviour interactions; via (ii) sloppy: occasionally imprecise representations, performances, event (re)actions, and behaviour interactions; and (iii) delinquent: repeatedly imprecise representations, performances, event (re)actions, and behaviour interactions; to (iv) criminal: outright counter productive, damaging representations, performances, event (re)actions, and behaviour interactions.

Abstraction of Human Behaviour. We extend the formalisation of rules and regulations.

Human actions (**ACT**) lead from a state (Θ) to any one of possible successor states (Θ**-infset**) — depending on the human behaviour, whether diligent, sloppy, delinquent or having criminal intent. The **hum**an **int**erpretation of a rule (**Rul**) usually depends on the current state (Θ) and can be any one of a possibly great number of semantic rules (**RUL**). For a delinquent (...) user the rule must yield truth in order to satisfy "being delinquent (...)".

type
  ACT = $\Theta \to \Theta$**-infset**
value
  hum_int: Rul $\to \Theta \to$ RUL**-infset**
  hum_behav: Sti $\times$ Rul $\to$ ACT $\to \Theta \to \Theta$**-infset**
  hum_behav(sti,rul)(α)(θ) **as** θs
    post θs = $\alpha(\theta)$ $\wedge$
      $\forall\ \theta'$:Θ • $\theta' \in \theta$s $\Rightarrow$
        $\exists$ se_rul:RUL • se_rul $\in$ hum_int(rul)(θ) $\Rightarrow$ se_rul(θ,θ')

Human **behav**iour is thus characterisable as follows: It occurs in a context of a **sti**mulus, a **rul**e, a present state (θ) and (the choice of) an action (α**:ACT**) which may have either one of a number of outcomes (θ**s**). Thus let θ**s** be the possible spread of diligent, sloppy, delinquent or outright criminal successor states. For each such successor states there must exist a rule interpretation which satisfies the pair of present an successor states. That is, it must satisfy being either diligent, sloppy, delinquent or having criminal intent and possibly achieving that!

Human Behaviour Suggested ℜesearch Topics:. Section 11.8 of Vol. 3 [3] elaborates on a number of ways of describing (i.e., modelling) human behaviour.

ℜ 11. **Concrete Methodology**: Based on the abstraction of human behaviour given earlier, one is to study how one can partition the set, $\alpha(\theta)$, of outcomes of human actions into 'diligent', 'sloppy', 'delinquent' and 'criminal' behaviours — or some such, perhaps cruder, perhaps finer partitioning — and for concrete cases attempt to formalise these for possible interactive "mechanisation".
ℜ 12. **Monitoring and Control of Human Behaviour**: Based on possible solutions to the previous research topic one is to study general such interactive "mechanisation" of the monitoring and control of human behaviour.

3.8 Domain Modelling: Suggested ℜesearch Topic

ℜ 13. **Sufficiency of Domain Facets**: We have covered five facets: intrinsics, support technology, management and organisation, rules and regulations and human behaviour. The question is: are these the only facets, i.e., views on the domain that are relevant and can be modelled? Another question is: is

there an altogether different set of facets, "cut up", so-to-speak, "along other lines of sights", using which we could likewise cover our models of domains?

One might further subdivide the above five facets (intrinsics, support technology, management and organisation, rules and regulations and human behaviour) into "sub"-facets. A useful one seems to be to separate out from the facet of rules and regulations the sub-facet of scripts.

• • •

We have finished our overview of domain facets.

4 Domains: Miscellaneous Issues

4.1 Domain Theories

– *By a* **domain theory** *we shall understand a domain description together with lemmas, propositions and theorems that may be proved about the description — and hence can be claimed to hold in the domain.*

To create a domain theory the specification language must possess a proof system. It appears that the essence of possible theorems of — that is, laws about — domains can be found in laws of physics. For a delightful view of the law-based nature of physics — and hence possibly also of man-made universes we refer to Richard Feynman's Lectures on Physics [4].

Example Theorem of Railway Domain Theory. Let us hint at some domain theory theorems: **Kirchhoff's Law for Railways:** Assume regular train traffic as per a modulo κ hour time table. Then we have, observed over a κ hour period, that the number of trains arriving at a station minus the number of trains ending their journey at that station plus the number of trains starting their journey at that station equals the number of trains departing from that station.

Why Domain Theories ? Well, it ought be obvious ! We need to understand far better the laws even of man-made systems.

Domain Theories: Suggested ℜesearch Topics:.

ℜ 14. **Domain Theories**: We need to experimentally develop and analyse a number of suggested theorems for a number of representative domains in order to possibly 'discover' some meta-theorems: laws about laws !

4.2 Domain Descriptions and Requirements Prescriptions

From Domains to Requirements. Requirements prescribe what "the machine", i.e., the hardware + software is expected to deliver. We show, in Vol. 3, Part V, Requirements Engineering, and in particular in Chap. 19, Sects. 19.4–19.5 how to construct, from a domain description, in collaboration with the requirements stakeholders, the domain (i.e., functional) requirements, and the interface (i.e., user) requirements.

Domain requirements are those requirements which can be expressed only using terms from the domain description. Interface requirements are those requirements which can be expressed only using terms from both the domain description and the machine — the latter means that terms of computers and software are also being used.

Domain requirements are developed as follows: Every line of the domain description is inspected by both the requirements engineer and the requirements stakeholders. For each line the first question is asked: *Shall this line of description prescribe a property of the requirements ?* If so it is "copied" over to the requirements prescription. If not it is "projected away". In similar rounds the following questions are then raised: *Shall the possible generality of the description be instantiated to something more concrete ? Shall possible non-determinism of the description be made less non-deterministic, more deterministic ? Shall the domain be "extended" to allow for hitherto infeasible entities, functions, events and behaviours ? Shall the emerging requirements prescription be "fitted" to elsewhere emerging requirements prescriptions ?* Similar "transformation" steps can be applied in order to arrive at (data initialisation and refreshment, GUI, dialogue, incremental control flow, machine-to-machine communication, etc.) interface requirements.

Domain and Interface Requirements: Suggested ℜesearch Topics.

ℜ 15. **Domain and Interface Requirements**: Vol. 3, Part V, Sects. 19.4–19.5 give many examples of requirements "derivation" principles and techniques. But one could wish for more research in this area: more detailed principles and techniques, on examples across a wider spectrum of problem frames.

4.3 Requirements-Specific Domain Software Development Models

A long term, that one: 'requirements-specific domain software development models' ! The term is explained next.

Software "Intensities". One can speak of 'software intensity'. Here are some examples. Compilers represent 'translation' intensity. 'Word processors', 'spread sheet systems', etc., represent "workpiece" intensity. Databases represent 'information' intensity. Real-time embedded software represent 'reactive' intensity. Data communication software represent connection intensity. Etcetera.

"Abstract" Developments. Let $''\mathcal{R}''$ denote the "archetypal" requirements for some specific software 'intensity'. Many different domains $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_i, \ldots, \mathcal{D}_j, \ldots\}$ may be subject to requirements $''\mathcal{R}''$-like prescriptions. For each such a set of possible software may result. The "pseudo-formula" below attempts, albeit informally, to capture this situation:

$$\left\{\begin{array}{l}\mathcal{D}_1\\ \mathcal{D}_2\\ \ldots\\ \mathcal{D}_i\\ \ldots\\ \mathcal{D}_k\\ \ldots\end{array}\right\} \sim\ ''\mathcal{R}''\ \mapsto\ \left[\begin{array}{c}\{\mathcal{S}_{1_1}, \mathcal{S}_{1_2}, \ldots, \mathcal{S}_{1_{j_1}}, \ldots\}\\ \{\mathcal{S}_{1_1}, \mathcal{S}_{1_2}, \ldots, \mathcal{S}_{1_{j_2}}, \ldots\}\\ \ldots\\ \{\mathcal{S}_{i_1}, \mathcal{S}_{i_2}, \ldots, \mathcal{S}_{i_{j_i}}, \ldots\}\\ \ldots\\ \{\mathcal{S}_{k_1}, \mathcal{S}_{k_2}, \ldots, \mathcal{S}_{k_{j_k}}, \ldots\}\\ \ldots\end{array}\right]$$

Several different domains, to wit: road nets and railway nets, can be given the "same kind" of (road and rail) maintenance requirements leading to information systems. Several different domains, to wit: road nets, railway nets, shipping lanes, or air lane nets, can be given the "same kind" of (bus, train, ship, air flight) monitoring and control requirements (leading to real-time embedded systems). But usually the specific requirements skills determine much of the requirements prescription work and especially the software design work.

Requirements-Specific Devt. Models: Suggested ℜesearch Topics.

ℜ 16_j. **Requirements-Specific Development Models, $\mathcal{RSDM}_j$:** We see these as grand challenges: to develop and research a number of requirements-specific domain (software) development models $\mathcal{RSDM}_j$.

The "pseudo-formal" $\prod(\sum_i^{\cdots} \mathcal{D}_i)\ ''\mathcal{R}''\ \sum_{i,j}^{\cdots} \mathcal{S}_{i_j}$ expression attempts to capture an essence of such research: The $\prod$ "operator" is intended to project (that is, look at only) those domains, $\mathcal{D}_i$, for which $''\mathcal{R}''$ may be relevant. The research explores the projections $\prod$, the possible $''\mathcal{R}''$s and the varieties of software $\sum_{i,j}^{\cdots} \mathcal{S}_{i_j}$.

4.4 On Two Reasons for Domain Modelling

Thus there seems to be two entirely different, albeit, related reasons for domain modelling: one justifies domain modelling on engineering grounds, the other on scientific grounds.

An Engineering Reason for Domain Modelling. In an e-mail, in response, undoubtedly, to my steadfast, perhaps conceived as stubborn insistence, on domain engineering, Sir Tony Hoare summed up his reaction, in summer of 2006, to domain engineering as follows, and I quote[5]:

"There are many unique contributions that can be made by domain modelling.

[5] E-Mail to Dines Bjørner, CC to Robin Milner et al., July 19, 2006

1. The models describe all aspects of the real world that are relevant for any good software design in the area. They describe possible places to define the system boundary for any particular project.
2. They make explicit the preconditions about the real world that have to be made in any embedded software design, especially one that is going to be formally proved.
3. They describe[6] the[7] whole range of possible designs for the software, and the whole range of technologies available for its realisation.
4. They provide a framework for a full analysis of requirements, which is wholly independent of the technology of implementation.
5. They enumerate and analyse the decisions that must be taken earlier or later in any design project, and identify those that are independent and those that conflict. Late discovery of feature interactions can be avoided."

All of these issues are dealt with, one-by-one, and in some depth, in Vol. 3 [3] of my three volume book.

A Science Reason for Domain Modelling. So, inasmuch as the above-listed issues of Sect. 4.4, so aptly expressed in Tony's mastery, also of concepts (through his delightful mastery of words), are of course of utmost engineering importance, it is really, in our mind, the science issues that are foremost: We must first and foremost understand. There is no excuse for not trying to first understand. Whether that understanding can be "translated" into engineering tools and techniques is then another matter. But then, of course, it is nice that clear and elegant understanding also leads to better tools and hence better engineering. It usually does.

Domains Versus Requirements-Specific Development Models. Sir Tony's five statements are more related, it seems, to the concept of requirements-specific domain software development models than to merely the concept of domain models. His statements help us formulate the research programme $\Re 16$ of requirements specific domain software development models. When, in his statements, you replace his use of the term 'models' with our term 'requirements-specific development models *based on domain models*', then "complete harmony" between the two views exists.

5 Conclusion

5.1 What Has Been Achieved ?

I set out to focus on what I consider the crucial modelling stage of describing domain facets and to identify a number of their research issues. I've done that.

[6] read: imply
[7] read: a

Cursorily, the topic is "near-holistic", so an overview is all that can be done. The issue is that of that of a comprehensive methodology. Hence the "holism" challenge.

5.2 What Needs to Be Achieved ?

Well, simply, to get on with that research. There are two sides to it: the 16 research topics mentioned above, and the ones mentioned below. The latter serves as a carrier for the former research.

Domain Theories: Grand Challenge ℜesearch Topics. The overriding research topic is that of:

ℜ 17_i. **Domain Models:** $\mathcal{D}_i$: We see this as a set of grand challenges: to develop and research a family of domain models $\mathcal{D}_i$.

5.3 Acknowledgements

I thank the organisers for inviting me to present a (this ?) talk. I thank UNU-IIST for inviting me and my wife back to Macau to a place where I spent great years. I consider UNU/IIST (as we spelled it in those days) one of my main achievements, so I also thank all those people who made it possible. They may have suffered then. But they too can be very proud now. I thank Sir Tony for fruitful discussions during the writing of this paper.

References

1. Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling.* Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
2. Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages.* Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen.
3. Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design.* Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
4. Richard Feynmann, Robert Leighton, and Matthew Sands. *The Feynmann Lectures on Physics*, volume Volumes I–II–II. Addison-Wesley, California Institute of Technology, 1963.