DOMAIN MODELLING

Dines Bjørner
Technical University of Denmark

Warszawa University of Technology, Tuesday 12 December 2023

The Triptych Dogma

In order to *specify* $\mathbb{S}$**oftware**,
we must understand its requirements.
In order to *prescribe* $\mathbb{R}$**equirements**
we must understand the domain
So we must **study, analyze** and **describe** $\mathbb{D}$**omains**.

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

1. DOMAINS

Definition 1 . ***Domain***

- By a *domain* we shall understand
  - a *rationally describable* segment of
  - a *discrete dynamics* fragment of
  - a *human assisted* reality, i.e., of the world.
- It includes
  - its **endurants**,
    i.e., *solid and fluid entities* of
    - **parts** and
    - **living species**,
  - and **perdurants** ■

- By **endurants** we shall understand
  - those quantities of domains
  - that we can observe (see and touch), in space,
  - as “complete” entities at no matter which point in time
  - “material” entities that persists, endures.
- By **perdurants** we shall understand an entity
  - for which only a fragment exists
  - if we look at or touch them at any given snapshot in time.
  - Were we to freeze time
  - we would only see or touch a fragment of the perdurant.

- *Endurants* are
  - either *natural* [“God-given”]
  - or *artefactual* [“man-made”].

  and may be considered
  - *atomic* or *compound* parts,
  - or, as in this talk, further unanalysed *living species:*
    - **plants** and
    - **animals** – including *humans*.
- *Perdurants* are here considered to be
  - *actions*,
  - *events* and
  - *behaviours*.
- *Perdurants* are **transcendentally deduced** from *parts.*

Example 1 . Domains: A few, more-or-less self-explanatory examples:

- **Rivers –** with their natural sources, deltas, tributaries, waterfalls, etc., and their man-made dams, harbours, locks, etc. – and their conveyage of materials (ships etc.) [19];
- **Road nets –** with street segments and intersections, traffic lights and automobiles – and the flow of these;
- **Pipelines –** with their wells, pipes, valves, pumps, forks, joins and wells and the flow of fluids [8]; and
- **Container terminals –** with their container vessels, containers, cranes, trucks, etc. – and the movement of all of these [15] ■

2. TWO LANGUAGE CLASSES

2.1 The Languages of Domains

- In naming specific or general instances of endurants
  - whether as a whole [i.e., their external qualities],
  - or their general or specific properties [i.e., their internal qualities],
  - we are normally using **nouns** of the “language” of the domain in question.
- In naming specific or general instances of perdurants
  - whether as a whole,
  - or their general or specific pretties,
  - we are normally using **verbs** of the “language” of the domain in question.
- That is, domain descriptions unveil **languages of domains,**
- i.e., are, in a sense **describing their semantics.**

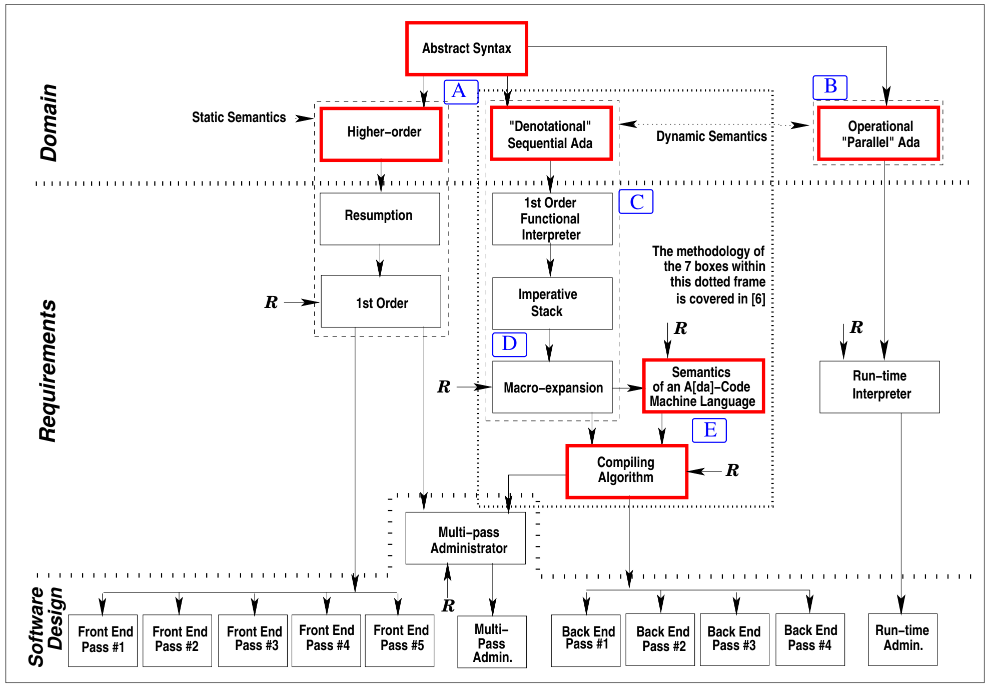


Fig. 2.1: The DDC CHILL and Ada compiler developments graph

2.2 The Languages of Programming

- We contrast that to the languages of programming.
  - For these we have the task of implementing interpreters and compilers.
  - Here is a *software development graph* for the development of compilers for languages such as CHILL and Ada.

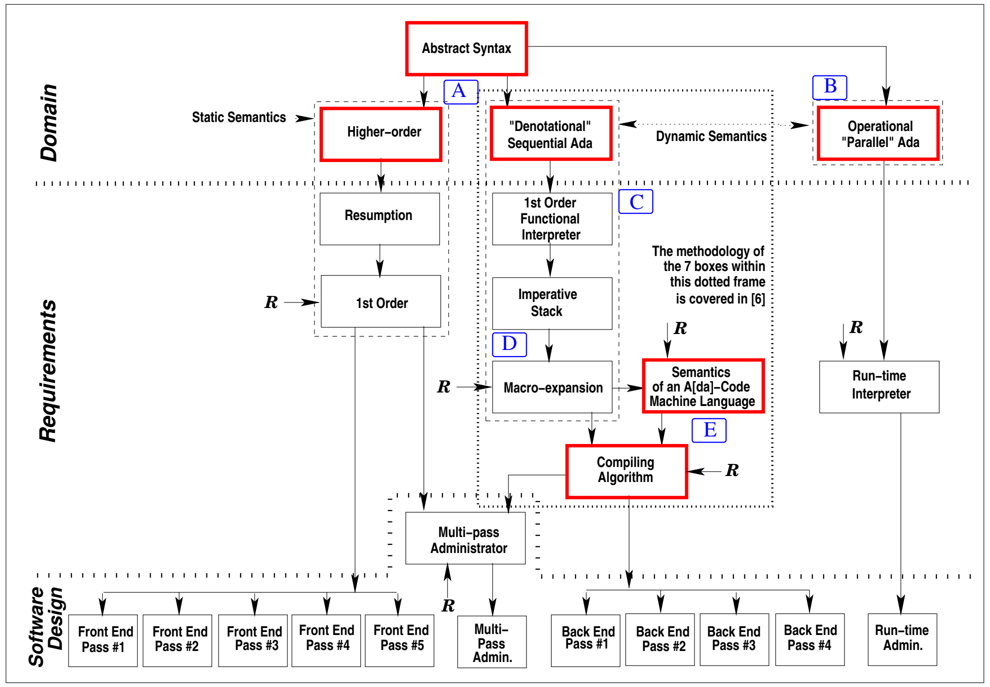


Fig. 2.2: The DDC CHILL and Ada compiler developments graph

- [A] Denotational Semantics, *McCarthy, Scott and Strachey*[40, 41, 45].
- [B] CSP, *Hoare*[34].
- [C] First Order Semantics, *Landin* and *Reynolds* [38, 43].
- [D] Imperative Stack and Macro-expansion Semantics, *Bekič* [1].
- [E] X Code to Compiling Algorithm, *McCarthy & Painter* [42].

2.3 A First Observation

- Of course we develop interpreters and compilers
  - for programming languages
  - by first describing their [static and dynamic] semantics.
- So, of course, we develop software
  - for any application domain,
  - by first describing its "semantics",
  - that is: a domain model.

• • •

- Engineers are intimately familiar with their natural science bases:
  - Telecommunications engineers with *Maxwell's Equations*.
  - Aircraft engineers with *Aero Dynamics*.
  - &c.

3. DOMAIN ANALYSIS & DESCRIPTION

3.1 The Natural and Man-made World Around Us !

- We shall focus on the artefactual world, made by us !
  - Some phenomena of that world we can explains, the **entities**,
  - some we cannot.
  - We shall focus on the *entities*.
  - We shall, in particular, focus on
    - ⋆ manifest **parts**, i.e., endurants, and their
    - ⋆ **behaviours**, i.e., perdurants.

- So how do we analyze & describe a[n application] domain ?
- Is there a **method**, principles, techniques, tools,
- for analysing & describing domains
  - Yes, basically as indicated by the ontology diagram:

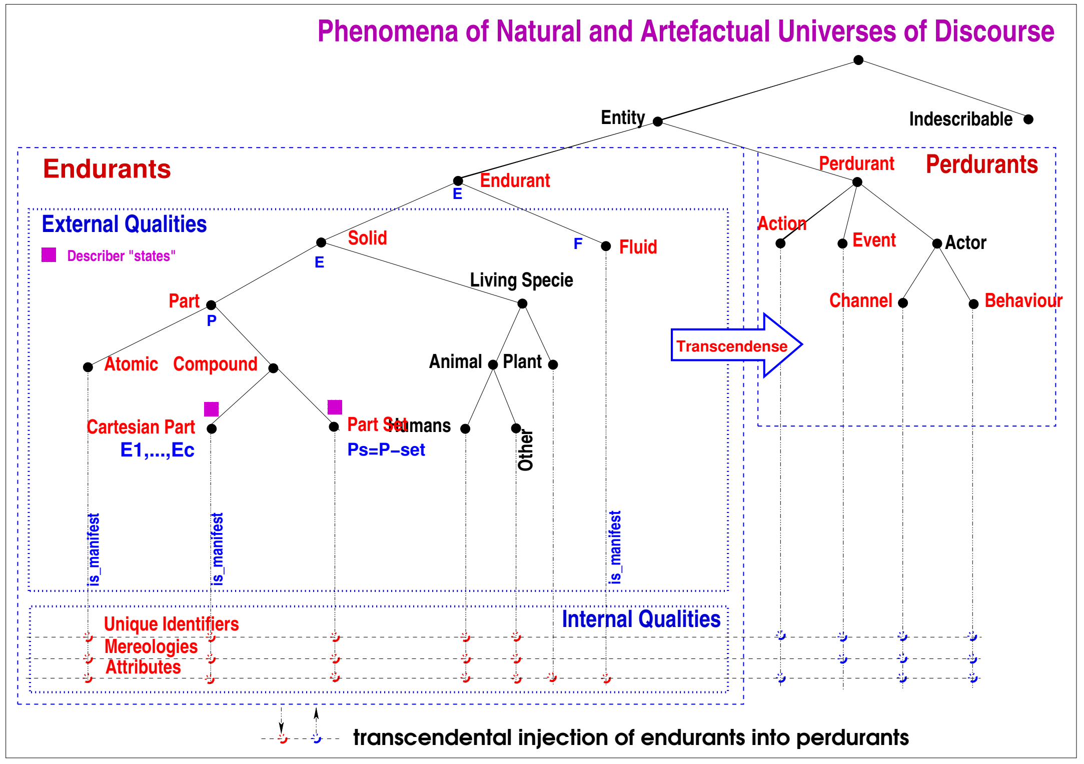


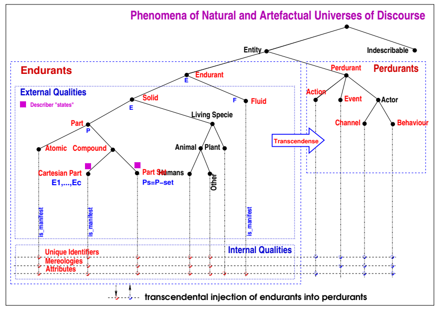
Fig. 3.1: A Domain Analysis & Description Ontology

3.2 Endurants and Perdurants

- There are two sides to the unfolding analysis & description:
  - The analysis & description of endurants, and
  - the analysis & description of perdurants.
- Within endurants there are further two sides:
  - The analysis & description of external qualities, those we can see and touch, and
  - the analysis & description of internal qualities those properties we can measure and/or speak about.
- And within the analysis & description of internal qualities there are the analysis & description of parts:
  - uniqueness,
  - mereology and
  - attributes.
- There is, finally, the concept of intentional pull.

3.3 Endurants: External Qualities

3.3.1 Analysis

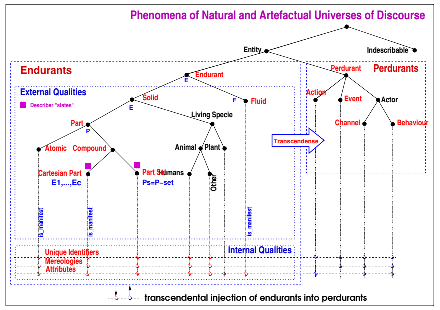


3.2: Our Domain Analysis & Description Ontology

We analyze what we see and ascertain:

- Endurant Ontology:
  - is_entity
  - is_endurant
  - is_perdurant
  - is_solid
  - is_fluid
  - is_part
  - is_living_species
  - is_atomic
  - is_compound
  - is_Cartesian
  - is_part_set
  - is_plant
  - is_animal
  - is_human
- Location:
  - is_stationary
  - is_mobile
- Treatment:
  - is_manifest
  - is_structure

3.3.2 Description



3.3: Our Domain Analysis & Description Ontology

- Auxiliary Description Functions:
  - determine_Cartesian_part_sorts
  - determine_part_set_sort
- Main description Functions::
  - calculate_Cartesian_part_sorts
  - calculate_part_set_sort

3.3.3 The Calculate Description Functions

A `calc_Cartesian_parts(e:E)` Schema

let (_,(ηE$_1$,...,ηE$_m$)) = determine_Cartesian_parts_sorts(e) **in**
“ **Narration:**
[s] ... narrative text on sorts ...
[o] ... narrative text on sort observers ...
[p] ... narrative text on proof obligations ...
Formalisation:
type
[s] E$_1$, ”...“ , E$_m$
value
[o] obs_E$_1$: E $\to$ E$_1$, ”...“ , obs_E$_m$: E $\to$ E$_m$
proof obligation
[p] [Disjointedness of endurant sorts] ”
end

A calc_part_set_sort(e:E) Schema

let (_,ηP) = determine_part_set_sort(e) **in**

❝ **Narration:**

[s] ... narrative text on sort ...

[o] ... narrative text on sort observer ...

[p] ... narrative text on proof obligation ...

Formalisation:

type

[s] P

[s] Ps = P-**set**

value

[o] obs_Ps: E → Ps ❞

proof obligation

[p] [Single sortness″ of Ps]❞

end

Example

1. The domain is that of a generic *road transport system*, RTS.
2. Of an RTS we can observe a Cartesian, `manifest` *road net aggregate*, RNA, and an *automobile aggregate*, AA, `structure`.
3. Of a *road net aggregate* we can observe a `structure` *set of hub aggregate*, HA, and a `structure` *set of link aggregate*, LA.
4. Of an HA we can observe a part set of atomic *hubs*, H, (i.e., street intersections).
5. Of an LA we can observe a part set of atomic *links*, L, (i.e., street segments).

Formalisation

type	value
1. RTS	2. obs_RNA: RTS→RNA
2. RNA, AA	2. obs_AA: RTS→AA
3. HA, LA	3. obs_HA: RNA→HA
4. Hs = H-**set**	3. obs_LA: RNA→LA
4. H	4. obs_Hs: HA→Hs
5. Ls = L-**set**	5. obs_Ls: LA→Ls
5. L	

3.4 Endurants: Internal Qualities

3.4.1 Unique Identifiers

- All entities are unique.
- Hence we associate unique identifiers with manifest parts.

A calc_unique_identifier Schema

❝ **Narration:**

[s] ... narrative text on unique identifier sort EI ...
[u] ... narrative text on unique identifier observer uid_E ...
[a] ... axiom on uniqueness of unique identifiers ...

Formalisation:

type
[s] UI
value
[u] uid_E: E → EI ❞

Example

6. Road net aggregates have unique identifiers.
7. Hubs have unique identifiers.
8. Links have unique identifiers.
9. They are all distinct.

Formalisation

type
6. RNI
7. HI
8. LI

value
6. uid_RN: RN→RNI
7. uid_H: H→HI
8. uid_L: L→LI

axiom
9. ∀ rn:RN ·
9. **let** his = {uid_H(h)|h∈obs_Hs(obs_HA(rn))},
9. lis = {uid_L(l)|l∈obs_Ls(obs_LA(rn))} **in**
9. uid_RN(rn) ∉ his ∪ lis ∧ his ∩ lis = {} **end**

3.5 An Aside: States

- For practical reasons, in the next formalizations, we need some state notions.
  - The manifest parts form a state, σ, and
  - The unique identifiers of manifest parts form a state, σ_{uid}.
- Given
  - a **unique identifier**, uid, in σ_{uid}
  - we can **retr**ieve the corresponding, unique **part** in σ.

Formalisation

value

rts:RTS

rna = obs_RNA(rts)

as = obs_As(obs_AA(rts))

hs = obs_Hs(obs_HA(obs_RNA(rts)))

ls = obs_Ls(obs_LA(obs_RNA(rts)))

σ = {rna} ∪ as ∪ hs ∪ ls

$rnai$ = uid_RNA(rna)

ais = { uid_A(a) | a:A • a ∈ as }

his = { uid_H(h) | h:H • h ∈ hs }

lis = { uid_A(a) | l:L • l ∈ ls }

σ_{uid} = {rna} ∪ ais ∪ his ∪ lis

retr_part: (RNI|AI|HI|LI) → Σ → P

retr_part(uid)(σ) ≡ **let** p:(RNA|A|H|L) • p ∈ σ ∧ uid_P(p)=uid **in** p **end**

3.5.1 Mereology

Definition 2 . ***Mereology:***
Mereology is the study and knowledge of parts and part relations ■

- Mereology was introduced in the form we shall use it by the Polish mathematician Staniław Leśniewski (1886–1939) [27, 9].
- Which are the relations that can be relevant for "endurant-hood"?
- There are basically two relations:
  - (i) spatial and
  - (ii) conceptual.

- (i) **Spatially** two or more endurants may be *topologically*
  - either adjacent to one another, like rails of a line,
  - or within an endurant, like links and hubs of a road net,
  - or an atomic part is conjoined to one or more fluids,
  - or a fluid is conjoined to one or more parts.
- (ii) **Conceptually** some parts, like automobiles,
  - “belong” to an embedding endurant,
    - ⋆ like to an automobile club, or
    - ⋆ are registered in the local department of vehicles,
  - or are ‘intended’ to drive on roads.

A calculate_mereology(p:P) Schema

❝ **Narration:**

[t] ... narrative text on mereology type ...
[m] ... narrative text on mereology observer ...
[a] ... narrative text on mereology type constraints ...

Formalisation:

type
[t] MT = $\mathcal{M}$(UI$_i$,UI$_j$,...,UI$_k$)
value
[m] mereo_P: P → MT
axiom [Well–formedness of Domain Mereologies]
[a] $\mathcal{A}$: $\mathcal{A}$(MT) ❞ ■

An Example

We shall only consider the mereologies of hubs, links and automobiles.

10. The mereology of a hub is a triple of the possibly empty set of the unique identifiers of the links that emanate from / are incident upon the hub, the automobiles that may enter the hub – and the road net unique identifier.
11. The mereology of a link is a triple of the exactly two element set of the unique identifiers of the hubs that are linked by the link, the automobiles that may enter the link– and the road net unique identifier.
12. The mereology of an automobile is the set of unique identifiers of the hubs and links that the automobile may enter.
13. All identifiers must be identifiers of the road transport system of the road net, hubs, links and automobiles; the link identifiers of a hub must be of links whose mereology prescribe those hubs; and, vice versa, the hub identifiers of a link must be of hubs whose mereology prescribe those links.

Formalisation

type
10. HM = LI-**set**×AI-**set**×RNI
11. LM = HI-**set**×AI-**set**×RNI
12. AM = (HI|LI)-**set**

value
10. mereo_H: H → HM
11. mereo_L: L → LM
12. mereo_A: A → AM

axiom
11. ∀ l:L · **let** (his,_,_)=mereo_L(l) **in card** his = 2 **end**
13. ∀ a:A,h:H,l:L · a ∈ as∧h ∈ hs∧l ∈ ls⇒
13. **let** hilis=mereo_A(a),
13. (lis′,ais′,rni′)=mereo_H(h),
13. (his″,ais″,rni″)=mereo_L(l) **in**
13. hilis⊆his∪lis
13. ∧ais′⊆ais∧ais″⊆ais
13. ∧lis′⊆lis∧lis″⊆lis
13. ∧his′⊆his∧his″⊆his
13. ∧rni′=rni∧rni″⊆rni[1] **end**

[1] For $as, hs, ls, rn, ais, his, lis$ and rni see Sect. 3.5 on Slide 22.

3.5.2 Attributes

- To recall: there are three sets of internal qualities:
  - unique identifiers,
  - mereologies and
  - attributes.
- Unique identifiers and mereologies
  are rather definite kinds of internal endurant qualities.
- Attributes form "wider-ranging" sets of internal qualities.

- We can roughly distinguish between two kinds of attributes:
  - those which can be motivated by **physical** (incl. chemical) **concerns**, and
  - those which, although they embody some form of 'physics measures', appear to reflect on **event histories**, i.e., **audit trails:**
    - *"if 'something', ϕ, has 'happened' to an endurant, e_a,*
    - *then some 'commensurate thing', ψ, has 'happened' to another (one or more) endurants, e_b."*
  - where the *'something'* and *'commensurate thing'*
  - usually involve some 'interaction' between the two (or more) endurants.
- It can take some reflection and analysis to properly identify
  - endurants e_a and e_b and
  - commensurate events ϕ and ψ.

A calculate_attributes Schema

let $\{\eta A_1, ..., \eta A_m\}$ = analyse_attribute_type_names(e) **in**

❝ **Narration:**

[t] ... narrative text on attribute sorts ...
some Ais may be concretely defined: [Ai=...]

[o] ... narrative text on attribute sort observers ...

[p] ... narrative text on attribute sort proof obligations ...

Formalisation:

type

[t] A_1[=...] , ..., A_m[=...]

value

[o] attr_A_1: E→A_1, ..., attr_A_m: E→A_m

Proof obligation [Disjointness of Attribute Types]

[p] $\mathcal{PO}$: is_A_i(a) ≠ is_A_j(a) [i≠j, i,j:[1..m]] ❞

end

Example

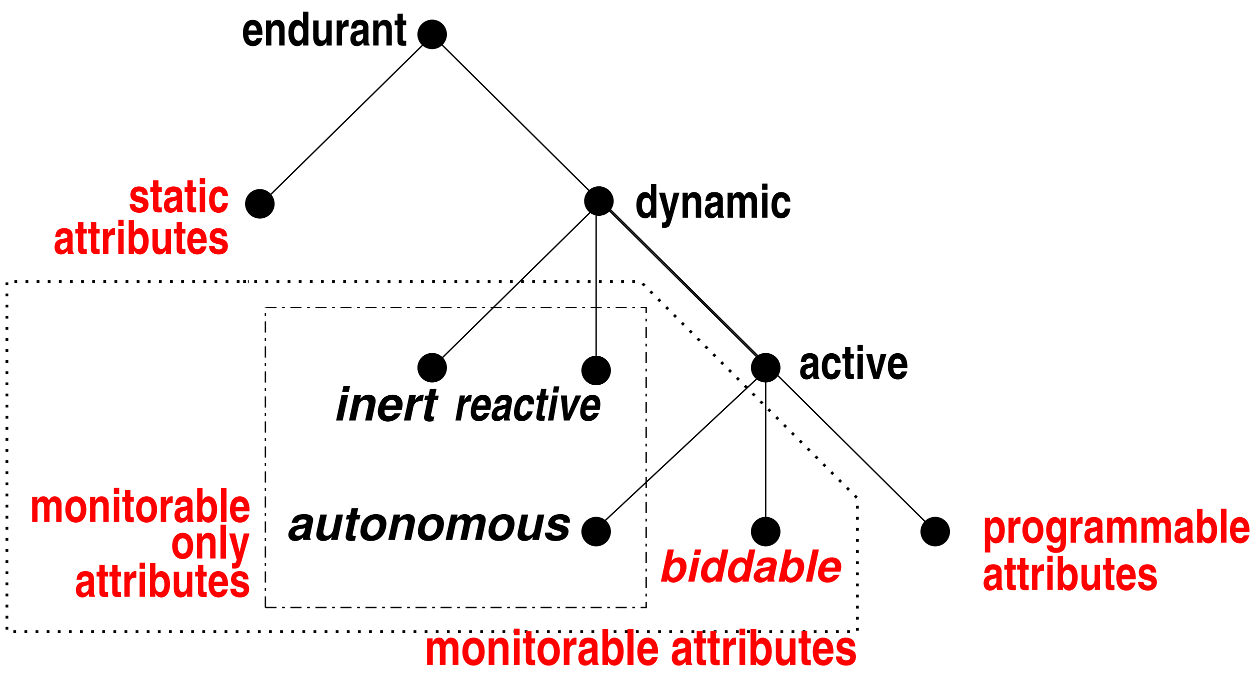
14. Hubs have signal states: possibly empty sets of pairs of incident link identifiers.
15. Hubs have state spaces: set of hub states.
16. And hubs have traffic histories: time-stamped automobile identifiers ... .
17. Links have traffic histories: time-stamped automobile identifiers ... .
18. Automobiles have location: either at a hub, from a link to a link, or a fraction down a link.
19. Automobile have traffic histories: time-stamped hub or link identifiers ... .

Formalisation

type
14. $\mathsf{H\Sigma = (LI{\times}LI)}$**-set**
15. $\mathsf{H\Omega = H\Sigma}$**-set**
16. $\mathsf{H\_Hist = (\mathbb{TIME}{\times}AI)^*}$
17. $\mathsf{L\_Hist = (\mathbb{TIME}{\times}AI)^*}$
18. A_Loc == at_a_Hub(fli:LI,hi:HI,tli:LI)
18. | on_a_Link(fhi:HI,li:LI,f:F,thi:HI)
18. F = **Real**, **axiom** 0<f<1
19. $\mathsf{A\_Hist = (\mathbb{TIME}{\times}(HI|LI))^*}$

value
14. attr_HΣ: H $\rightarrow$ HΣ
15. attr_HΩ: H $\rightarrow$ HΩ
16. attr_H_Hist: H $\rightarrow$ H_Hist
17. attr_L_Hist: L $\rightarrow$ L_Hist
18. attr_A_Loc: A $\rightarrow$ A_Loc
19. attr_A_Hist: A $\rightarrow$ A_Hist

Michael A. Jackson's Attribute Categories



3.4: Attribute Value Ontology [36]

3.5.3 Intentional Pull

- *Intentionality*
  - "expresses" conceptual, abstract relations
  - between otherwise, or seemingly unrelated entities.

- The Oxford English Dictionary [39] characterizes intentionality as follows:
  - *the quality of mental states*
    (e.g. thoughts, beliefs, desires, hopes)
  - *which consists in their being directed*
  - *towards some object or state of affairs.*

Informal Examples

We present three examples.

- **Automobile Traffic:**
  - If an automobile history "records" being on a road or link
  - at time τ,
  - then that road or link must "record" the presence of that automobile
  - at that time;

  AND:
  - If a hub or link history "records" an automobile at time τ,
  - then that automobile must "record" its presence on that hub, respectively link
  - at that time ■

- **Double-entry Bookkeeping:**
  - The outlay/expense sum total
  - must balance
  - the active/passive sum total ■

- **The Henry George Theorem:**
  - The Henry George theorem states that
  - under certain conditions, aggregate spending
  - by government on public goods
  - will increase aggregate rent based on land value (land rent)
  - more than that amount,
  - with the benefit of the last marginal investment equaling its cost ■

3.6 Perdurants

3.6.1 Transcendental Deduction

Some Definitions

Definition 3 . ***Transcendental:***
By transcendental
we shall understand the philosophical notion:
the a priori or intuitive basis of knowledge,
independent of experience ■

Definition 4 . ***Transcendental Deduction:***
By a transcendental deduction
we shall understand the philosophical notion:
a transcendental "conversion"
of one kind of knowledge
into a seemingly different kind of knowledge ■

Examples

- Trains, in a service center,
  being maintained,
  can be considered *endurants.*
- Those same trains, now in operation,
  "speeding" down the rail tracks,
  can, by transcendental deduction, be considered *perdurants.*
- And: "trains" referred to in time-tables
  can be considered *time-table attributes*.

3.7 Morphing Endurants into Perdurants: Parts into Behaviours

- Thus, to every **endurant part**
  we shall associate, by *transcendental deduction*,
  a **perdurant behaviour**.

3.8 Analysis of Perdurants

- *Part behaviours* are characterized by
  - **actions** pertaining to the individual part behaviours and
  - **events** pertaining to the (**channel**) *interaction*
    between part behaviours – with
  - part behaviours "alternating", non-deterministically,
    externally ($\square$) or internally ($\sqcap$),
    between two or more actions and/or two or more events.
- We shall describe these behavioral issues using CSP [34].

3.9 Description Details

3.9.1 Channel Description

- So behaviours interact via channels.
- In general any two [part] behaviours may communicate.
- So we consider the channels to be a double-indexed array of simple channels:

channel { ch[{ui,uj}] | ui,uj:PI · of any domain parts }

Example

- The channel array of the road transport system.

Formalisation

channel { ch[{ui,uj}] | ui,uj:(RNI|AI|HI|LI) · ui≠uj ∧ {ui,uj}⊆σ_{uid} }

3.9.2 Actions and Events I

- Actions pertain to one or more behaviours.
- Actions **are planned**
  and may change the state of its related behaviours.
- Events pertain to one behaviour.
- Events **are not planned,** but **occur surreptitiously**
  and may change the state of its behaviour.

Examples

- **Actions:**
  - An automobile remains in a hub.
  - An automobile remains on a link.
  - An automobile leaves a hub and enters a link.
  - An autombile leaves a link and enters a hub.
  - An autombile exits the road net.
- **Events:**
  - An automobile ceases to be an automobile[2].
  - A link, for example a tunnel or a bridge, breaks down[3].

[2] motor breaks down, or crashes
[3] fire, mud slide, or other

3.9.3 Behaviour Signatures

- schematic form of part (p) behaviour signatures is:

  b: bi:BI→me:Mer→svl:StaV*→mvl:MonV*→prl:PrgV* channels **Unit**

  - b: name of part p behaviour
  - bi: p unique identifier
  - mer: p mereology
  - svl: p static attributes
  - mvl: p monitorable attributes
  - prl: p programmable attrs.
  - channels: subset of channels
  - **Unit:** the () state value

Example: Behaviour Signatures

20. automobile:
    (a) unique identifier, mereology, static (...) and monitorable (...) attributes;
    (b) programmable attributes: automobile location and automobile history;
    (c) and channel references
    allowing communication between the automobile and the hub and link behaviours.
21. Similar for link and hub behaviours.

Formalisation

```
value
20a automobile: ai:AI → (_,uis):AM → ... → ...
20b      → (A_Loc × A_Hist)
20c        out {ch[ {ai,ui} ]|ui:(HI|LI) • ui ∈ his∪lis} Unit
21a   hub: hi:HI → (lis,ais,rni):HM → (HΩ × ...) → ...
21b      → (HΣ×H_Hist)
21c        in {ch[ {hi,ui} ]|ui:(I|HI|RNI)-set • ui ∈ lis∪lis∪rni} Unit
21a   link: li:LI → (his,ais,rni):LM → (LEN × LΩ × ...)
21b      → (LΣ×L_Hist)
21c        in {ch[ {li,ui} ]|ui:(I|HI|RNI)-set • li ∈lis∪his∪rni} Unit
```

3.9.4 Behaviour Invocation

b(bi)(me)(svl)(mvl)(prl)

3.9.5 Behaviour and Action Definition Schemes

Behaviours

```
behaviour(bi)(me)(svl)(mvl)(prl) ≡
      nd_action_1(bi)(me)(svl)(mvl)(prl)
   ⊓ nd_action_2(bi)(me)(svl)(mvl)(prl)
      ...
   ⊓ nd_action_n(bi)(me)(svl)(mvl)(prl)
   ⌷ d_action_1(bi)(me)(svl)(mvl)(prl)
   ⌷ d_action_2(bi)(me)(svl)(mvl)(prl)
      ...
   ⌷ d_action_d(bi)(me)(svl)(mvl)(prl)
```

Actions

```
action(bi)(me)(svl)(mvl)(prl) ≡
     let prl′ = act(bi)(me)(svl)(mvl)(prl) in behaviour(bi)(me)(svl)(mvl)(prl′) end
```

Example: Behaviour Definitions

22. We abstract automobile behaviour at a Hub (hi).
23. Internally non-deterministically, an automobile either
24. either progresses around the hub
25. or leaves the hub to enter a link.

Formalisation

```
22 automobile(ai)(aai,uis)(...)(apos:at_a_Hub(fli,hi,tli),ahist) ≡
24    automobile_progress_around_hub(ai)(aai,uis)(...)(a_loc:at_a_Hub(fli,hi,tli),ahist)
23       ⊓
25    automobile_leave_hub_enter_link(ai)(aai,uis)(...)(a_loc:at_a_Hub(fli,hi,tli),ahist)
```

26. The automobile progresses around the hub:
    (a) the automobile at that hub,
    (b) informing ("first") the hub behaviour.

Formalisation

```
26  automobile_progress_around_hub(ai)(aai,uis)(...)(at_a_Hub(fli,hi,tli),ahist) ≡
26      let τ = record_TIME() in
26b      ch[ai,hi] ! τ ;
26a      automobile(ai)(aai,uis)(...)(at_a_Hub(fli,hi,tli),upd_hist(τ,hi)(ahist))
26      end

26a  upd_hist: (TIME×UI) → (A_Hist→A_Hist)|(H_Hist→H_Hist)|(L_Hist→L_Hist)
26a  upd_hist(τ,ui)(hist) ≡ hist † [ui ↦ ⟨τ⟩⁀hist(ui)]
```

27. The automobile leaves the hub entering a link:
    (a) tli, whose "next" hub, identified by thi, is obtained from the mereology of the link identified by tli;
    (b) informs the hub it is leaving and the link it is entering,
    (c) "whereupon" the vehicle resumes (i.e., "while at the same time" resuming) the vehicle behaviour positioned at the very beginning (0) of that link.

Formalisation

27 automobile_leave_hub_enter_link(ai)(aai,uis)(...)(a_loc:at_a_Hub(fli,hi,tli),ahist) ≡
27a (**let** ({fhi,thi},ais) = mereo_L(retr_L(tli)(σ)) **in assert:** fhi=hi
27b (ch[ai,hi] ! τ ‖ ch[ai,tli] ! τ) ;
27c automobile(ai)(aai,uis)(...)(on_a_Link(tli,(hi,thi),0),upd_hist(τ,tli)(upd_hist(τ,hi)(ahist))) **end**)

28. Or the automobile "disappears — off the radar" !

Formalisation

28 automobile_stop(ai)(aai,uis),(...)(apos:atH(fli,hi,tli),ahist) ≡ **stop**

3.9.6 Domain Instantiation

- For every manifest part sort
  - there is a single description: signature and definition (i.e., its syntax).
- For every manifest part
  - there is a behaviour (i.e., its semantics "realization").
- For the total of all manifest domain parts there is their initialization:
  - the parallel "execution"
  - of the behaviour of each manifest part,
  - properly initialized.

Example: Domain Initialization

29. Let us refer to the system initialization as a behaviour.
30. All links are initialized,
31. all hubs are initialized,
32. all automobiles are initialized,
33. etc.

Formalisation

value
29. rts_initialisation: **Unit** → **Unit**
29. rts_initialisation() ≡
30. ‖ { link(uid_L(l))(mereo_L(l))(attr_LEN(l),attr_LΩ(l))(attr_L_Traffic(l),attr_LΣ(l))| l:L • l ∈ *ls* }
31. ‖ ‖ { hub(uid_H(l))(mereo_H(l))(attr_HΩ(l))(attr_H_Traffic(l),attr_HΣ(l))| h:H • h ∈ *hs* }
32. ‖ ‖ { automobile(uid_A(a))(mereo_A(a))(attr_RegNo(a))(attr_APos(a)) | a:A • a ∈ *as* }
33. ‖ ...

4. SUMMING UP !

4.1 What has [Not] been Achieved ?

4.1.1 Achieved

- We have outlined a method —
  - hinting at its principles, procedures, techniques and tools —
  - for analyzing and describing a certain class of domains.
- This method “heralds” an extension within software development:
  - before there was *requirements engineering* and *software design,*
  - now *domain engineering,* and its *science*, is prefixed that approach.

4.1.2 Not Achieved

- The next-but-following section of the talk hints at some open issues.

4.2 Experimental Domain Models

- The domain analysis & description method, its
  - principles,
  - procedures,
  - techniques and
  - tools
- have been honed over many years
- through domain modelling experiments – some are:

- *railways* [2, 23, 4],
- *"The Market"* [3],
- *container shipping* [5],
- *Web systems* [6],
- *stock exchange* [7],
- *oil pipelines* [8],
- *credit card systems* [11],
- *weather information* [12],
- *swarms of drones* [13],
- *document systems* [14],
- *container terminals* [15],
- *retail systems* [17],
- *assembly plants* [16],
- *waterway systems* [19],
- *shipping* [20], and
- *urban planning* [26].

4.3 Open Issues

4.3.1 The Rôle of Algorithms

- Which rôle[1] do algorithms play in all this ?
  - Basically no role !
  - We describe properties.
  - Not how to compute properties.

4.3.2 RSL, The RAISE Specification Language

- We use a slight extension of the RAISE [32] Specification Language, RSL [31].
- We could as well have used similar extensions to either of
  - VDM [24, 25, 29],
  - Alloy [35],
  - cafeOBJ [30],
  - or other !

[1]– or just role !

4.3.3 Continuity

- To model the behaviour of discrete dynamic domains,
  - such as are the main focus of this talk,
  - we use the CSP process concept [34].
- To model the behaviour of continuous dynamic domains,
  - which we really have not,
  - we suggest that You use methods of classical analysis,
  - to wit: *[Partial] Differential Equations, PDEs*.
  - Perhaps also some *Fuzzy Logic* [48, 37].
- That is: We see this as the “dividing line” between
  - discrete and
  - continuous

  dynamic systems modelling: *CSP versus PDEs*.
- But: Current formal, logic-based specification languages do not mesh easily with classical calculi !
- See, however, [46, 47, *rTiMo, BigrTiMo*].

4.3.4 Domain Facets – covered in [18] Chapter 8

.

- 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.
- As examples of domain facets we list
  - intrinsics,
  - support technologies,
  - rules & regulations,
  - scripts,
  - license languages,
  - management & organization, and
  - human behaviour.

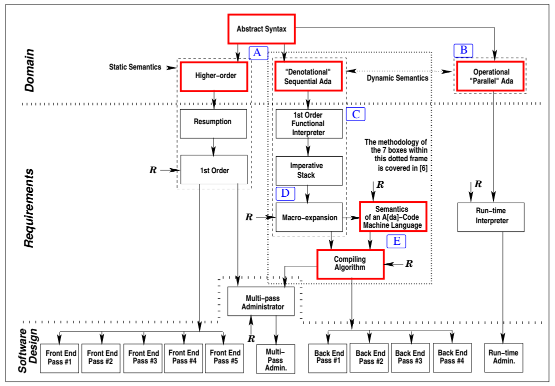
  as such facets.

4.3.5 Requirements Engineering– covered in [18] Chapter 9

- We shall view requirements from three "sides":
  - (α) *domain requirements*,
  - (β) *interface requirements*, and
  - (γ) *machine requirements*.
- But first a definition of the term *'machine'*.
  - By machine we shall understand
    - a, or the, combination of hardware and software
    - that is the target for, or result of
    - the required computing systems development.
- By a *requirements* we shall understand (cf., IEEE Standard 610.12):
  - *"A condition or capability*
  - *needed by a user*
  - *to solve a problem*
  - *or achieve an objective."* ■

- By a *domain requirements* we shall understand
  - those requirements
  - which can be expressed
  - sôlely using terms of the domain ■
- By an *interface requirements* we shall understand
  - those requirements
  - which can be expressed
  - only using technical terms of both the domain and the machine ■
- By a *machine requirements* we shall understand
  - those requirements which, in principle,
  - can be expressed
  - sôlely using terms of the machine ■

- The *domain requirements* stage of requirements development
  - starts with a basis in the domain engineering's domain description.
  - It is, so-to-speak, a first step in the development of a requirements prescription.
  - From there follows, according to [18, Chapter 9] a number of (five) steps:
    - (1.) *projection*
    - (2.) *instantiation*
    - (3.) *determination*
    - (4.) *extension*
- The *interface and machine requirements* stages of requirements development can be decomposed "similarly"!



4.1: The DDC CHILL and Ada compiler developments graph

4.3.6 From Programming to Domains

Theories of Compiler Development

- Trustworthy compiler development is based on many theories; to wit:
  - [A] Denotational Semantics, *McCarthy, Scott and Strachey*[40, 41, 45].
  - [B] CSP, *Hoare*[34].
  - [C] First Order Semantics, *Landin* and *Reynolds* [38, 43].
  - [D] Imperative Stack and Macro-expansion Semantics, *Bekič* [1].
  - [E] *X* Code to Compiling Algorithm, *McCarthy & Painter* [42].

- A trustworthy progress, from "top" to "bottom" of the above diagram reflects Unifying Theories of Programming [33].

Domain Specific Languages

- A domain specific language, generically referred to as a DSL,
  - is a language whose basic syntactic elements directly reflect endurants and perdurants of a specific domain.
  - *Actulus*, a language in which to express calculations of actuarian character [28], is a DSL.
- The semantics of a DSL, obviously, must relate to a model for the domain in question.
- In fact, we advice, that DSLs be developed from the basis of relevant domain models.

- A guiding rule for the development of DSLs is their adherence to *The Dogma of* Unifying Theories of Programming

Philosophy of Computing

- The Danish philosopher Kai Sørlander
  - has shown that there is a foundation in philosophy
  - for domain analysis and description.
- We refer to [18, *Chapter 2*] for a summary of his findings.

4.3.7 Possible PhD Topics

- Domain science & engineering offers scientific challenges:
  - Domain Specific Languages & Unifying Theories of Programming
  - Role of $\mathbb{D},\mathbb{S} \models \mathbb{R}$ in Program Verification
  - Intentional Pull
  - Continuous Behaviours [46, 47, *rTiMo, BigrTiMo*]
  - Towards a Calculus of Perdurants
  - Modelling Human Interaction
  - Further Study of Domain Facets [18, Chapter 8]
  - Further Study of Domain Requirements [18, Sect. 9.4]
  - Further Study of Interface Requirements [18, Sect. 9.5]
  - Formalizing Domain Calculi [10]
  - Transcendental Deduction
  - Kai Sørlander's Philosophy

5. ACKNOWLEDGMENTS

- Prof. Andrzej Jacek Blikle, IPIPAN
- Prof. Andrzej Tarlecki, MIM, UW
- Prof. Krzysztof Brzezinski, WUoT

THANKS!

Bibliography

[1] H. Bekič and C. B. Jones, editors. *Programming Languages and Their Definition: Selected Papers of H. Bekič*, volume 177 of *Lecture Notes in Computer Science*. Springer-Verlag, 1984.

[2] Dines Bjørner. Formal Software Techniques in Railway Systems. In Eckehard Schnieder, editor, *9th IFAC Symposium on Control in Transportation Systems*, pages 1–12, Technical University, Braunschweig, Germany, 13–15 June 2000. VDI/VDE-Gesellschaft Mess– und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug– und Verkehrstechnik. Invited talk.

[3] Dines Bjørner. 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)*, The Netherlands, December 2002. Kluwer Academic Press. www2.imm.dtu.dk/˜dibj/themarket.pdf.

[4] Dines Bjørner. Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In *CTS2003: 10th IFAC Symposium on Control in Transportation Systems*, Oxford, UK, August 4-6 2003. Elsevier Science Ltd. Symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki. www2.imm.dtu.dk/˜dibj/ifac-dynamics.pdf.

[5] Dines Bjørner. A Container Line Industry Domain. www.imm.dtu.dk/ db/container-paper.pdf. Techn. report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, June 2007.

[6] Dines Bjørner. On Development of Web-based Software: A Divertimento of Ideas and Suggestions. Technical, Technical University of Vienna, August–October 2010. www.imm.dtu.dk/˜dibj/wfdftp.pdf.

[7] Dines Bjørner. The Tokyo Stock Exchange Trading Rules www.imm.dtu.dk/˜db/todai/tse-1.pdf, www.imm.dtu.dk/˜db/todai/tse-2.pdf. R&D Experiment, Techn. Univ. of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, 2010.

[8] Dines Bjørner. Pipelines – a Domain www.imm.dtu.dk/˜dibj/pipe-p.pdf. Experimental Research Report 2013-2, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Spring 2013.

[9] Dines Bjørner. A Rôle for Mereology in Domain Science and Engineering. In *Mereology and the Sciences*, Synthese Library (eds. Claudio Calosi and Pierluigi Graziani), pages 323–357, Amsterdam, The Netherlands, October 2014. Springer.

[10] Dines Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model www.imm.dtu.dk/ dibj/2014/kanazawa/kanazawa-p.pdf. In Shusaku Iida and José Meseguer and Kazuhiro Ogata, editor, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, Heidelberg, Garmany, May 2014.

[11] Dines Bjørner. A Credit Card System: Uppsala Draft www.imm.dtu.dk/˜dibj/2016/credit/accs.pdf. Technical Report: Experimental Research, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, November 2016.

[12] Dines Bjørner. Weather Information Systems: Towards a Domain Description www.imm.dtu.dk/˜dibj/2016/wis/wis-p.pdf. Technical Report: Experimental Research, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, November 2016.

[13] Dines Bjørner. A Space of Swarms of Drones. www.imm.dtu.dk/˜dibj/2017/swarms/swarm-paper.pdf. Research Note, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, December 2017.

[14] Dines Bjørner. What are Documents ? www.imm.dtu.dk/˜dibj/2017/docs/docs.pdf. Research Note, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, July 2017.

[15] Dines Bjørner. Container Terminals. www.imm.dtu.dk/ dibj/2018/yangshan/maersk-pa.pdf. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, September 2018. An incomplete draft report; currently 60+ pages.

[16] Dines Bjørner. *An Assembly Plant Domain – Analysis & Description*, `www.imm.dtu.dk/˜dibj/2021/assembly/assembly-line.pdf`. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, September 2019.

[17] Dines Bjørner. A Retailer Market: Domain Analysis & Description. A Comparison Heraklit/DS&E Case Study. www.imm.dtu.dk/ dibj/2021/Retailer/BjornerHeraklit27January2021.pdf. Technical Report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, January 2021.

[18] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, Heidelberg, Germany, 2021. A revised version of this book is [22].

[19] Dines Bjørner. Rivers and Canals. www.imm.dtu.dk/˜dibj/2021/Graphs/Rivers-and-Canals.pdf. Technical Report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, March 2021.

[20] Dines Bjørner. Shipping. www.imm.dtu.dk/˜dibj/2021/ral/ral.pdf. Technical Report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, April 2021.

[21] Dines Bjørner. Domain Modelling – A Primer. A short version of [22]. xii+202 pages[1], May 2023.

[22] Dines Bjørner. Domain Science & Engineering – A Foundation for Software Development. Revised edition of [18]. xii+346 pages[2], January 2023.

[23] Dines Bjørner, Chris W. George, and Søren Prehn. 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*, P.O.Box 1299, Grand View, Texas 76050-1299, USA, 24–28 June 2002. Society for Design and Process Science. www2.imm.dtu.dk/˜dibj/pasadena-25.pdf.

[24] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, Heidelberg, Germany, 1978.

[25] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, London, England, 1982.

[26] Dines Bjørner. Urban Planning Processes. www.imm.dtu.dk/˜dibj/2017/up/urban-planning.pdf. Research Note, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, July 2017.

[27] Roberto Casati and Achille C. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.

[28] David R. Christiansen, Klaus Grue, Henning Niss, Peter Sestoft, and Kristján S. Sigtryggsson. Actulus Modeling Language - An actuarial programming language for life insurance and pensions. Technical Report, edlund.dk/sites/-default/files/Downloads/paper_actulus-modeling-language.pdf, Edlund A/S, Denmark, Bjerregårds Sidevej 4, DK-2500 Valby. (+45) 36 15 06 30. edlund@edlund.dk, http://www.edlund.dk/en/insights/scientific-papers, 2015. This paper illustrates how the design of pension and life insurance products, and their administration, reserve calculations, and audit, can be based on a common formal notation. The notation is human-readable and machine-processable, and specialised to the actuarial domain, achieving great expressive power combined with ease of use and safety.

[29] John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.

[30] K. Futatsugi, A.T. Nakagawa, and T. Tamai, editors. *CAFE: An Industrial–Strength Algebraic Formal Method*, Sara Burgerhartstraat 25, P.O. Box 211, NL–1000 AE Amsterdam, The Netherlands, 2000. Elsevier. Proceedings from an April 1998 Symposium, Numazu, Japan.

[31] Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[32] Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Development Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

[33] C.A.R. Hoare and Ji Feng He. *Unifying Theories of Programming*. Prentice Hall, 1997.

[34] Charles Anthony Richard Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, London, England, 1985. Published electronically: usingcsp.com/cspbook.pdf (2004).

[35] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

[36] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England, 1995.

[1] This book is currently being translated into Chinese by Dr. Yang ShaoFa, IoS/CAS, Beijing and into Russian by Dr. Mikhail Chupilko, ISP/RAS, Moscow

[2] Due to copyright reasons no URL is given to this document's possible Internet location. A primer version, omitting certain chapters, is [21]

[37] James J. Buckley and Esfanidar Eslami. *An Introduction to Fuzzy Logic and Fuzzy Sets*. Springer, 2002.

[38] Peter J Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.

[39] W. Little, H.W. Fowler, J. Coulson, and C.T. Onions. *The Shorter Oxford English Dictionary on Historical Principles*. Clarendon Press, Oxford, England, 1973, 1987. Two vols.

[40] John McCarthy. Recursive Functions of Symbolic Expressions and Their Computation by Machines, Part I. *Communications of the ACM*, 3(4):184–195, 1960.

[41] John McCarthy. Towards a Mathematical Science of Computation. In C.M. Popplewell, editor, *IFIP World Congress Proceedings*, pages 21–28, 1962.

[42] John McCarthy and James Painter. Correctness of a Compiler for Arithmetic Expressions. In *[44]*, pages 33–41, 1966. Dept. of Computer Science, Stanford University, California, USA.

[43] John C Reynolds. Definitional interpreters for higher-order programming languages. In *Proceedings of the ACM annual conference-Volume 2*, pages 717–740. ACM, 1972.

[44] Jack T. Schwartz. *Mathematical Aspects of Computer Science, Proc. of Symp. in Appl. Math.* American Mathematical Society, Rhode Island, USA, 1967.

[45] Dana S. Scott and Christopher Strachey. Towards a mathematical semantics for computer languages. In *Computers and Automata*, volume 21 of *Microwave Research Inst. Symposia*, pages 19–46, 1971.

[46] WanLing Xie, ShuangQing Xiang, and HuiBiao Zhu. A UTP approach for rTiMo. *Formal Aspects of Computing*, 30(6):713–738, 2018.

[47] WanLing Xie, HuiBiao Zhu, and Xu QiWen. A process calculus BigrTiMo of mobile systems and its formal semantics. *Formal Aspects of Computing*, 33(2):207–249, March 2021.

[48] Lotfi A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

5.1: **Left:** The gypsum model, Thorvaldsens Museum, Copenhagen
Right: My grandfather, 1911, Modlin Castle, Homel, Count Paszkiewicz.