Towards a Theory of Domain Descriptions
— Bergen 8 May Mini-course Notes —

Dines Bjørner
DTU Informatics, Techn.Univ.of Denmark, DK–2800 Kgs.Lyngby
Fredsvej 11, DK-2840 Holte, Denmark
bjorner@gmail.com, www.imm.dtu.dk/~db

May 1, 2012: 16:29

Contents

1 Introduction 6
1.1 Rôles of Domain Engineering . . . 6
1.1.1 Software Development . . . 6
Requirements Construction . . . 6
Software Design . . . 6
1.1.2 Domain Studies "In Isolation" . . . 7
1.2 Additional Preliminary Notions . . . 7
1.2.1 Types and Values . . . 7
1.2.2 Algebras . . . 9
Abstract Algebras . . . 9
Heterogeneous Algebras . . . 9
Behavioral Algebras . . . 10
1.3 On 'Method' and 'Methodology' . . . 14
1.4 An Ontology of Descriptions . . . 15
1.4.1 Entities and Properties . . . 15
1.4.2 Categories of Entities . . . 15
1.5 Structure of Paper . . . 15

2 Domains 16
2.1 Informal Characterisation . . . 16
2.2 Mereology . . . 16
2.3 Rough Sketch Hints of Domains . . . 17
2.4 What are Domains ? . . . 18
2.4.1 An Informal Characterisation of Domains . . . 18
2.4.2 A Formal Characterisation of Domains . . . 18
2.5 Six Examples . . . 19
2.5.1 Air Traffic . . . 19
2.5.2 Buildings . . . 19
2.5.3 Financial Service Industry . . . 20
2.5.4 Machine Assemblies . . . 20

2.5.5 Oil Industry . . . 21
"The" Overall Assembly . . . 21
A Concretised Composite parts . . . 22
2.5.6 Railway Nets . . . 23

3 Entities 25
Examples . . . 25
3.1 Parts . . . 26
Examples . . . 26
3.1.1 Atomic Parts . . . 26
Examples . . . 26
3.1.2 Composite Parts . . . 27
Examples . . . 27
3.1.3 Part Attributes . . . 27
Atomic Part Attributes . . . 28
Examples . . . 28
Composite Part Attributes . . . 28
Examples . . . 28
Static Part Attributes . . . 29
Examples . . . 29
Dynamic Part Attributes . . . 29
Examples . . . 29
Indivisibility of Attributes . . . 30
Examples . . . 30
3.1.4 Subparts Are Parts . . . 32
Examples . . . 32
3.1.5 Subpart Types Are Not Subtypes . . . 32
Examples . . . 32
3.1.6 Mereology of Composite Parts . . . 32
Examples . . . 32
3.1.7 Part Descriptions . . . 33
Examples . . . 34
3.1.8 States . . . 34
Examples . . . 34
3.2 Actions . . . 35
Examples . . . 35
3.3 Events . . . 36
Example . . . 36
3.4 Behaviours . . . 37
Example . . . 37
3.5 Discussion . . . 38

4 Describing Domain Entities 39
4.1 On Describing . . . 39
4.1.1 Informal Descriptions . . . 39
Domain Instances Versus Domains . . . 39
Non-uniqueness of Domain Descriptions . . . 39
A Criterion for Description . . . 39
Reason for 'Description' Failure . . . 40
Failure of Description Language . . . 40
Guidance . . . 40
4.1.2 Formal Descriptions . . . 40

4.2 A Formal Description Language . . . 41
4.2.1 Observing and Describing Entities . . . 41
4.2.2 Observing and Describing Parts . . . 41
Abstract Types . . . 41
Concrete Types . . . 42
Type Definitions . . . 42
Type Properties . . . 44
Subpart Type Observers . . . 45
Unique Identifier Functions . . . 45
Mereologies and Their Functions . . . 46
General Attributes and Their Functions . . . 47
4.2.3 Describing Actions . . . 49
Function Names . . . 49
Informal Function Descriptions . . . 49
Formal Function Descriptions . . . 50
Agents . . . 52
4.2.4 Describing Events . . . 52
Deliberate and Inadvertent (Internal and External) Events . . . 52
Event Predicates . . . 52
4.2.5 Describing Behaviours . . . 54
Behaviour Description Languages . . . 54
Simple Sequential Behaviours . . . 54
— *Snapshot Description of a Simple Sequential Behaviour:* . . . 54
Simple Concurrent Behaviours . . . 55
Communicating Behaviours . . . 55
External Non-deterministic Behaviours . . . 55
Internal Non-deterministic Behaviours . . . 56
General Communicating Behaviours . . . 56
4.3 Temporal Issues . . . 61
4.3.1 Three Abstract Time Concepts . . . 61
4.3.2 Concrete Time Concepts . . . 61
4.3.3 Some Interval Relations . . . 62
4.3.4 Time Phenomena . . . 62
Parts and Time . . . 62
Actions and Time . . . 63
Events and Time . . . 63
Behaviours and Time . . . 64
4.3.5 Temporal Descriptions . . . 67

5 Discovering Domain Entities 68
5.1 Preliminaries . . . 68
5.1.1 Part Signatures . . . 68
5.1.2 Domain Indices . . . 68
5.1.3 Inherited Domain Signatures . . . 69
5.1.4 Domain and Sub-domain Categories . . . 69
5.1.5 Simple and Compound Indexes . . . 69
5.1.6 Simple and Compound Domain Categories . . . 70
5.1.7 Examples . . . 70
5.1.8 Discussion . . . 74
5.2 Proposed Type and Signature 'Discoverers' . . . 75
5.2.1 Analysing Domain Parts . . . 76

Domain Part Sorts and Their Observers . . . 76
A Domain Sort Discoverer . . . 76
Domain Part Types and Their Observers . . . 77
Do a Sort Have a Concrete Type ? . . . 77
A Domain Part Type Observer . . . 78
Concrete Part Types . . . 79
Part Type Analysers . . . 79
Unique Identity Analysers . . . 79
Mereology Analysers . . . 79
General Attribute Analysers . . . 80
— *Attribute Sort Exploration* . . . 82
5.2.2 Discovering Action Signatures . . . 82
General . . . 82
Function Signatures Usually Depend on Compound Domains . . . 82
The ACTION_SIGNATURES Discoverer . . . 82
5.2.3 Discovering Event Signature . . . 83
5.2.4 Discovering Behaviour Signatures . . . 83
5.3 What Does Application Mean ? . . . 85
5.3.1 PART_SORTS . . . 86
5.3.2 HAS_A_CONCRETE_TYPE . . . 86
5.3.3 PART_TYPES . . . 86
5.3.4 UNIQUE_ID . . . 87
5.4 Discussion . . . 87

6 Conclusion **89**
6.1 General . . . 89
6.2 What Have We Achieved ? . . . 89
6.3 Other Formal Models . . . 89
6.4 Research Issues . . . 89
6.5 Engineering Issues . . . 89
6.6 Comparable Work . . . 89
6.7 Acknowledgements . . . 89

7 Bibliographical Notes **90**
7.1 The Notes . . . 90
7.2 References . . . 90

8 Indexes **93**
8.1 Index of Concepts . . . 93
8.2 Index of Examples . . . 95
8.3 Index of Formulas . . . 96
2 8.4 Index of Inquieries . . . 101

Abstract

We seek foundations for a possible theory of domain descriptions. Sect. 2 informally outlines what we mean by a domain. Sect. 3 informally outlines the entities whose description form a description of a domain. Sect. 4 then suggests one way of formalising such description parts[1]. There are other ways of formally describing

[1]The exemplified description approach is model-oriented, specifically the `RAISE` [23] cum `RSL` [22] approach.

domains[2], but the one exemplified can be taken as generic for other description approaches. Sect. **??** outlines a theory of domain mereology. Sect. 5 suggests some 'domain discoverers'. 3

These research notes reflect our current thinking. Through seminar presentations, their preparation and post-seminar revisions it is expected that they will be altered and honed.

[2]Other model-oriented approaches are those of `Alloy` [28], `Event B` [1], `VDM` [7, 8, 17] and `Z` [42]. Property-oriented description approaches include `CafeOBJ` [19], `Casl` [13] and `Maude` [32, 12]

1 Introduction 4

In this section we shall cover a number of concepts ("Preliminary Notions" and "An Ontology of Descriptions", Sects. 1.2–1.4) that lie at the foundation of the theory and practice of domain science and engineering. These are general issues such as (i) software engineering as consisting of domain engineering, requirements engineering, and software design, (ii) types and values, and (iii) algebras. But first we shall put the concept of domain engineering in a proper perspective.

1.1 Rôles of Domain Engineering 5

By domain engineering we shall understand the engineering[3] of domain descriptions, their study, use and maintenance. In this section (Sect. 1.1) we shall focus on the use of domain descriptions (i) in the construction of requirements and, from these, in the design of software, and (ii) more generally, and independent of requirements engineering and software design, in the study of man-made domains in a search for possible laws.

1.1.1 Software Development 6

We see domain engineering as a first in a triptych phased software engineering: (I) domain engineering, (II) requirements engineering and (III) software design. Sections 3–4 cover some engineering aspects of domain engineering.

7

Requirements Construction As shown elsewhere [3, 4, 5, 6] domain descriptions, $\mathcal{D}$, can serve as a firm foundation for requirements engineering. This done is by systematically "deriving" major part of the requirements from the domain description. The 'derivation' is done in steps of refinements and extensions. Typical steps reflect such 'algebraic
8 operations' as projection, instantiation, determination, extension, fitting, etcetera In "injecting" a domain description, $\mathcal{D}$, in a requirements prescription, $\mathcal{R}$, the requirements engineer endeavors to satisfy *goals*, $\mathcal{G}$, where goals are meta-requirements, that is, are a kind of higher-order requirements which can be uttered, that is, postulated, but cannot be formalised in a way from which we can "derive" a software design. For the concept of 'goal' we refer to [30, Axel van Lamsweerde].

So, to us, domain engineering becomes an indispensable part of software engineering. In [6] we go as far as suggesting that current requirements engineering (research and practice) rests on flawed foundations !

9

Software Design Finally, from the requirements prescription, $\mathcal{R}$, software, $\mathcal{S}$, can be designed through a series of refinements and transformations such that one can prove

[3]Engineering is the discipline, art, skill and profession of acquiring and applying scientific, mathematical, economic, social, and practical knowledge, in order to design and build structures, machines, devices, systems, materials and processes . . . [http://en.wikipedia.org/wiki/Engineering]

$\mathcal{D}, \mathcal{S} \models \mathcal{R}$, that is, the software design, $\mathcal{S}$, models, i.e., is a correct implementation of the requirements, $\mathcal{R}$, where the proof makes assumptions about the domain, $\mathcal{D}$.

1.1.2 Domain Studies "In Isolation" 10

But one can pursue developments of domain descriptions whether or not one subsequently wishes to pursue requirements and software design. Just as physicists study "mother nature" in order just to understand, so domain scientists cum engineers can study, for example, man-made domains — just to understand them. Such studies of man-made domains 11
seem worthwhile. Health care systems appear to be quite complex, embodying hundreds or even thousands of phenomena and concepts: parts, actions, events and behaviours. So do container lines, manufacturing, financial services (banking, insurance, trading in securities instruments, etc.), liquid and gaseous material distribution (pipelines), etcetera. Proper studies of each of these entails many, many years of work.

1.2 Additional Preliminary Notions 12

We first dwell on the "twinned" notions 'type' and 'value', Sect. 1.2.1. And then we summarise, Sect. 1.2.2, the notions of (universal, or abstract) algebras, heterogeneous algebras and 'behavioural' algebras. The latter notion, behavioural algebra, is a "home-cooked" term. (Hence the single quotes.) The algebra section, Sect. 1.2.2, is short on definitions and long on examples.

1.2.1 Types and Values 13

Values $(0, 1, 2, \ldots)$ have types (**integer**). We observe values (**false, true**)), but we speak of them by their types (**Bool**ean); that is: types are abstract concepts whereas (actual) values are (usually) concrete phenomena. By a type we shall here, simplifying, mean a way of characterising a set of entities (of similar "kind"). Entity values and types are related:when we observe an entity we observe its value; and when we say that an entity is of a given type, then we (usually) mean that the observed entity is but one of several entities *of that type.* 14

Example 1 (Types and Values of Parts) Three naïve examples

When we say, or write, *the* [or *that*] *net*, we mean

1. an entity, a specific value, n,
2. of type *net*, N.

When we say, or write, *the* [or *that*] *account*, we mean

3. an entity, a specific value, a,
4. of type *account*, A.

When we say, or write, *the* [or *that*] *container*, we mean

5. an entity, a specific value, c,
6. of type *container*, C.

type	**type**	**type**
2. N	4. A	6. C
value	**value**	**value**
1. n:N	3. a:A	5. c:C

•

15

Example 2 (Types and Values of Actions, Events and Behaviours) We continue the example above: A set of actions that all insert hubs in a net have the common signature:

value
  insert: $H \to N \overset{\sim}{\to} N$

The type expression $H{\to}N\overset{\sim}{\to}N$ demotes an infinite set of functions from Hubs to partial functions from Nets

to Nets. The **value** clause insert: $H{\to}N\overset{\sim}{\to}N$ names a function value in that infinite set insert and non-deterministically selects an arbitrary value in that infinite set. The functions are partial ($\overset{\sim}{\to}$) since an argument Hub may already "be" in the N in which case the insert function is not defined. A set of events that all result in a link of a net being broken can be characterised by the same predicate signature: 16

value
  link_disappearance: $N \times N \to$ **Bool**

The set of behaviours that focus only on the insertion and removal of hubs and links in a net have the common signature:

type
  Maintain = Insert_H | Remove_H | Insert_L | remove_L
value
  maintain_N: $N \to \text{Maintain}^* \to N$
  maintain_N: $N \to \text{Maintain}^\omega \to$ **Unit**

If insertions and removals continue ad infinitum, i.e., $^\omega$, then the maintenance behaviour do likewise: **Unit**.

•

Inquiry: Type and Value

The concept of type and its study in the last 50 years, is, perhaps, the finest contribution that computer science have made to mathematics. It all seems to have started with Bertrand Russel who needed to impose a type hierarchy on sets in order to understand the problem posed by the question: "is the set of all sets a member of itself". Explicit types were (one may claim) first introduced into programming languages in `Algol 60` *[2].*

The two concepts: 'type' and 'value' go hand-in-hand. $\boxed{\text{MORE TO COME}}$ •

1.2.2 Algebras 17

Abstract Algebras By an *abstract algebra* we shall understand a (finite or infinite) set of parts (e_1, e_2, ...) called the *carrier*, A (a type), of the algebra, and a (usually finite) set of functions, f_1, f_2, ..., f_n, [each] in Ω, over these. Writing $f_i(e_{j_1}, e_{j_2}, \ldots, e_{j_m})$, where f_i is in Ω of signature:

$$\textbf{signature}\,\omega : A^n \;\rightarrow\; A$$

and each e_{j_ℓ} (ℓ : $\{1..m\}$) is in A. The operation $f_i(e_{j_1}, e_{j_2}, \ldots, e_{j_m})$ is then meant to designate either **chaos** (a totally undefined quantity) or some e_k in A.

Heterogeneous Algebras A *heterogeneous algebra* has its carrier set, A, consist of a number of usually disjoint sets, also referred to as sub-types of A: A_1, A_2, ..., A_n, and a set of operations, ω:Ω, such that each operation, ω, has a *signature*: 18

$$\textbf{signature}\,\omega : A_i{\times}A_j{\times}\cdots{\times}A_k \;\rightarrow\; A_r$$

where $A_i, A_j, \ldots, A_k$ and A_r are in $\{A_1, A_2, \ldots, A_n\}$. 19

Example 3 (Heterogeneous Algebras: Platoons) We leave it to the reader to fill in missing narrative and to decipher the following formalisation.

7. There are vehicles.
8. A platoon is a set of one or more vehicles.

type
7. V
8. P = {| p • p:V-**set** ∧ p≠{} |}

9. A vehicle can join a platoon.
10. A vehicle can leave a platoon.
11. Two platoons can be merged into one platoon.
12. A platoon can be split into two platoons.

20

9. join_0: V × P → P
9. join_0(v,p) ≡ p ∪ {v} **pre**: v ∉ p

10. leave_0: V × P → P
10. leave_0(v,p) ≡ p\{v} **pre**: v ∈ p

11. merge_0: P × P → P

11. merge_0(p,p′) $\equiv$ p $\cup$ p′ **pre**: p $\neq$ {} $\neq$ p′ $\wedge$ p $\cap$ p′ = {}

12. split_0: P $\rightarrow$ P-**set**
12. split_0(p) $\equiv$ **let** p′,p″:P • p′ $\cup$ p″ = p **in** {p′,p″} **end pre**: **card** p $\geq$ 2

The above formulas define a heterogeneous algebra with types V and P and operations (or actions) *join_0*, *leave_0*, *merge_0*, and *split_0*.

●

21

Behavioral Algebras An abstract algebra is characterised by the one type, A, of its parts and by its operations all of whose signatures are of the form $A \times A \times \cdots \times A \rightarrow A$. A heterogeneous algebra is an abstract algebra and is further characterised by two or more types, A_1, A_2, $\ldots$, A_m, and by a set of operations of usually distinctly typed signatures. A behavioral algebra is a heterogeneous algebra and is further characterised by a set of events and by a set of behaviours where events are like actions and behaviours are sets of sequences of actions, events and behaviours. 22

Example 4 (**A Behavioural Algebra: A System of Platoons and Vehicles**) Our example may be a bit contrived. We have yet to unfold, as we do in this paper, enough material to give more realistic examples.

13. A well-formed platoon/vehicle system consists of a pair:
    a *convoys* which is a varying set of [non-empty] platoons and
    b *reservoir* which is a varying set of vehicles —
    c such that the *convoys* platoons are disjoint, no vehicles in common, and
    d such that *reservoir* have no vehicle in common with any platoon in *convoys*.

23

14. Platoons are characterised by unique platoon identifiers.
15. These identifiers can be observed from platoons.
16. Vehicles from the *reservoir* behaviour may join [leave] a platoon whereby they leave [respectively join] the pool.
17. Two platoons may merge into one, and a platoon may split into two.
18. Finally, vehicles may enter [exit] the system by entering [exiting] *reservoir*.

24

type
13. S = {| (c,r):C×R•r $\cap$ $\cup$ c = {} |}
13a. C = {| c:P-**set** • wf_C(c) |}
value

```
13c.  wf_C: C → Bool
13c.  wf_C(c) ≡ ∀ p,p′:P•{p,p′}⊆c ⇒ p≠{}≠p′ ∧ p ∩ p′ = {}
type
13b.  R = V-set
value
16.    join_1:  S ~→ S
16.    leave_1:  S ~→ S
17.    merge_1: S ~→ S
17.    split_1: S ~→ S
18.    enter_1: S ~→ S
18.    exit_1: S ~→ S
```

25

19. join_1 selects an arbitrary vehicle in *r:R* and an arbitrary platoon *p* in *c:C*, joins *v* to *p* in *c* and removes *v* from *r*.

20. leave_1 selects a platoon *p* in *c* and a vehicle *v* in *p*, removes *v* from *p* in *c* and joins *v* to *r*.

21. merge_1 selects two distinct platoons *p*,*p′* in *c*, removes them from *c*, takes their union and adds to *c*.

22. split_1 selects a platoon *p* in *c*, one which has at least to vehicles,

23. and partitions *p* into *p′* and *p″*, removes *p* from *c* and joins *p′* and *p″* to *c*.

24. enter_1 joins a fresh vehicle *v* to *r*.

25. exit_1 removes a vehicle *v* from a non-empty *r*.

26

```
19.  join_1(c,r) ≡
19.     let v:V•v ∈ r,p:P•p ∈ c in
19.     (c\{p} ∪ {join_0(v,p)},r\{v}) end

20.  leave_1(c,r) ≡
20.     let v:V,p:P•p ∈ c ∧ v ∈ p in
20.     (c\{p} ∪ {leave_0(v,p)},r ∪ {v}) end

21.  merge_1(c,r) ≡
21.     let p,p′:P•p≠p′∧{p,p′}⊆c in
21.     (c\{p,p′} ∪ {merge_0(p,p')},r) end

22.  split_1(c,r) ≡
23.     let p:P•p ∈ c∧card p≥2 in
```

23.   **let** p',p'':P•p $\cup$ $p' = p$ **in**
23.   (c$\backslash${p} $\cup$ **split_0(p)**,r) **end end**

24. enter_1(c,r) $\equiv$ (c,**let** v:V•v$\notin$ r $\cup$ $\cup$ c **in** r $\cup$ {v} **end**)
25. exit_1()(c,r) $\equiv$ (c,**let** v:V•v $\in$ r **in** r$\backslash${v} **end**) **pre**: r$\neq${}

The r $\cup$ $\cup$c in enter_1(c,r) expresses the union (with the vehicles of r) of all the vehicles in all the platoons of c, i.e., the distributed union of c ($\cup$c). 27

The above model abstracts an essence of the non-deterministic behaviour of a platooning system. We make no assumptions about which vehicles are joined to or leave which platoons, which platoons are merged, which platoon is split nor into which sub-platoons, and which vehicle enters and exits the reservoir state. 28

26. We model the above *system* as a behaviour which is composed from a pair of concurrent behaviours:

    a a *convoys* behaviour and

    b a *reservoir* behaviour

    c where these behaviours interact via a channel *cr_ch* and

    d where the entering of "new" and exiting of "old" vehicles occur on a channel *io_ch*

27. Hence the communications between the *reservoir* behaviour and the *convoys* behaviour are of three kinds: *J*oining (moving) a vehicle to a ("magically"[4]) named platoon from the *reservoir* behaviour, *R*emoving [moving] a vehicle from a named platoon to (*mkV(v)*) the *reservoir* behaviour

29

type
27.   M == mkJ(v:V) | mkR | mkV(v:V)
channel
26c.   cr_ch:M
26d.   io_ch:V
value
26.   system: S $\rightarrow$ **Unit**
26.   system(c,r) $\equiv$ convoys(c) $\parallel$ reservoir(r)

30

28. The *convoys* behaviour non-deterministically ($\sqcap$) chooses either to

    a merge platoons, or to

[4]In this example we skip the somewhat 'technical' details as to how the *reservoir* behaviour obtains knowledge of platoon names.

b split platoons, or to

c interact with the *reservoir* behaviour via channel *ct_ch*

d and based on that interactions

  i. to either join a[n arbitrary] vehicle *v* to a platoon, or

  ii. to remove a named vehicle, *v*, from a platoon

  iii. while “moving’ that vehicle to *reservoir*.

31

```
28.       convoys: C → in,out cr_ch  Unit
28.       convoys(c) ≡ convoys(merge(c)) ⌈⌉ convoys(split(c)) ⌈⌉ convoys(interact(c))

28c.      interact: C → in,out cr_ch  C
28c.      interact(c) ≡
28c.          let m = cr_ch ? in
28d.          case m of
28(d)i.           mkJ(v) → join_vehicle(v,c),
28(d)ii.          mkR  → let (c′,v)=remove_vehicle(c) in
28(d)iii.                    ct_ch!mkV(v) ; c′
28c.          end end end
```

32

29. The *merge_platoons* behaviour

  a non-deterministically chooses two platoons of *convoys* (p,p'),

  b removes the two platoons from *convoys* and adds the *merge* of these two platoons to *convoys*.

  c If *convoys* contain less than two platoons then *merge_platoons* is undefined.

```
29.       merge_platoons: C → C
29.       merge_platoons(c) ≡
29a.          let p,p′,p″:P • p≠p′∧{p,p′}⊆ c in
29b.          c\{p,p′} ∪ {merge_0(p,p’)} end
29b.          pre: card c ≥ 2
```

33

30. The *split_platoons* function

  a non-deterministically chooses a platoon, *p*, of two or more vehicles in *convoys*,

  b removes the chosen platoon from *convoys* and inserts the split platoons into *convoys*.

  c If there are no platoons in *c* with two or more vehicles then *split_platoons* is undefined.

```
30.   split_platoons: C $\xrightarrow{\sim}$ C
30.   split_platoons(c) $\equiv$
30a.      **let** p:P • p $\in$ c $\wedge$ **card** p $\geq$ 2 **in**
30b.      c\{p} $\cup$ {**split_0(p)**} **end**
30c.      **pre**: $\exists$ p:P • p $\in$ c $\wedge$ **card** p $\geq$ 2
```

34

31. The *reservoir* behaviour interacts with the *convoys* behaviour and with "an external", that is, undefined behaviour through channels *ct_ch* and *io_ch*.
    The *reservoir* behaviour [external] non-deterministically chooses between
    - a importing a vehicle from "the outside",
    - b exporting a vehicle to "the outside",
    - c moving a vehicle to the *convoys* behaviour, and
    - d moving a vehicle from the *convoys* behaviour.

35

```
31.   reservoir: R → **in**,**out** cr_ch, io_ch  **Unit**
31.   reservoir(r) ≡
31a.        (r ∪ {io_ch?}),
31b.      ⌷ **let** v:V • v ∈ t **in** io_ch!mkV(v) ; reservoir(r\{v}) **end**
31c.      ⌷ **let** v:V • v ∈ t **in** ct_ch!mkJ(v) ; reservoir(r\{v}) **end**
31d.      ⌷ **let** mkV(v) = ct_ch? **in** reservoir(r ∪ {v}) **end**
```

We may consider Items 31a–31b as designating events.

This example designates a behavioural algebra.

•

Inquiry: Algebra

Algebra is a mathematical notion. We shall use this notion in seeking to describe domains as algebras.

$\boxed{\text{MORE TO COME}}$

•

1.3 On 'Method' and 'Methodology' 36

Inquiry: Method and Methodology

We present our characterisation of the concepts of 'method' and 'methodology'. When we use these terms then our characterisation is what we mean by their use. There are other characterisations. Be that as it may. •

By a *method* we shall understanda set of *principles*, *techniques* and *tools* where the principles help *select* and *apply* these techniques and tools such that an *artifact*, here a domain description, can be constructed.

By *methodology* we shall understand the *knowledge* and *study* of one or more methods.

Languages, whether informal, as English, or formal, as RSL, are *tools*.

1.4 An Ontology of Descriptions 37

"By ontology we mean the philosophical study of the nature of being, existence, or reality as such, as well as the basic categories of being and their relations. Traditionally listed as a part of the major branch of philosophy known as metaphysics, ontology deals with questions concerning what entities exist or can be said to exist, and how such entities can be grouped, related within a hierarchy, and subdivided according to similarities and differences."[5]

1.4.1 Entities and Properties 38

A main stream of philosophers [34, 20, 18] appear to agree that there are two categories of discourse: entities[6] and properties. Once we say that, a number of questions arise: (Q_1) What counts as an entity ? (Q_2) What counts as a property ? (Q_3) Are properties entities ? (Q_4) Can properties predicate properties ? We shall take no and yes to be answers to Q_3 and Q_4. These lecture notes shall answer Q_1 and Q_2

1.4.2 Categories of Entities 39

We shall promulgate the following classes of entities: parts, and operations. where we further "sub-divide" operations into actions, events and behaviours That is, we can predicate entities, e, as follows: $\mathbb{IS\_PART}(e)$, $\mathbb{IS\_OPERATION}(e)$, that is, $\mathbb{IS\_ACTION}(e)$, $\mathbb{IS\_EVENT}(e)$ and $\mathbb{IS\_BEHVAIOUR}(e)$. We shall justify the above categorisation through these lecture notes. So parts, actions, events and behaviours form an ontology of descriptions.

1.5 Structure of Paper 40

1. Introduction 6–15
2. Domains 16–24
3. Entities 25–38
4. Describing Domain Entities 39–67
   a Parts, Actions, Events 39–54
   b Behaviours 54–67
5. Discovering Domain Entities 68–88
6. Conclusion 89–89

[5]http://en.wikipedia.org/wiki/Ontology

[6]The literature [31, 10, 11, 34, 20, 18, 41] alternatively refer to entities by the term individuals.

2 Domains 41

By an observable phenomenon we shall here understand something that can be sensed by one or more of our five sense organs. By a domain we shall here informally understand an area of human activity characterised by observable phenomena: entities and their properties, and abstractions, i.e., concepts, thereof. In Sect. 2.4 we suggest a more formal way of characterising a domain. But first we give some rough sketch hints as to what domains are.

Inquiry: Rough-sketching and Rough Sketch

> *We shall be using the idea of 'rough-sketching' descriptions (prescriptions and specifications) as a means to give the reader a rough, but not yet sufficiently precise idea of what we are aiming at. Rough-sketching (as a verb) a domain description is a process which helps the 'rough-sketcher' in first discovering the parts, actions, events and behaviours of a domain. Rough sketch (as a noun) is the result of rough-sketching and serves to help the rough-sketcher to formulate (not the, but) an essence.* •

2.1 Informal Characterisation 42

There are several forms of observable phenomena. There are the entities: **endurant**[7] entities: **parts**, and **perdurant**[8] entities: **actions**, **events**, and **behaviours** of the domain. Then there are the properties of these entities: (i) their **unique identifications**, (ii) the **mereology** of parts, that is, how parts are "put together", parts within, or subparts of other parts, etcetera, and (iii) the **attributes** of parts: types and values, whether atomic or composite, and of actions, events and behaviours: signatures and values. We will just 43 examine one of the part properties.

2.2 Mereology

Mereology, to us, is the study and knowledge about how physical and conceptual parts relate and what it means for a part to be related to another part: being *adjacent* to, being 44 *contained* properly within, being *overlapped* (i.e., *sharing*) properly with, etcetera.

By physical parts we mean such spatial individuals which can be pointed to. **Examples:** *a road net (consisting of street segments and street intersections); a street segment (between two intersections); a street intersection; a vehicle; and a platoon (of sequentially* 45 *adjacent vehicles).*

By a conceptual part we mean an abstraction with no physical extent, which is either present or not. **Examples:** *a bus timetable (not as a piece or booklet of paper, or as an electronic device, but) as an image in the minds of potential bus passengers; and routes of a pipeline, that is, adjacent sequences of pipes, valves, pumps, forks and joins, for example*

[7]Endurants are those entities that can be observed-perceived as a complete concept, at no matter which given snapshot of time [Wikipedia].

[8]Perdurants are those entities for which only a part exists if we look at them at any given point in time. When we freeze time we can at most see a part of the perdurant. [Wikipedia].

referred to in discourse: take "such-and-such" a route". The tricky thing here is that a route may be thought of as being both a concept or being a physical part — in which case one ought give them different names: a planned route and an actual route, for example. 46

The mereological notion of subpart, that is: *contained within* can be illustrated by **examples:** *the intersections and street segments are subparts of the road net; vehicles are subparts of a platoon; and pipes, valves, pumps, forks and joins are subparts of pipelines.* The mereological notion of adjacency can be illustrated by **examples:** *the pipes of a pipeline are adjacent (that is, connected) to other pipes or valves or pumps or forks or joins, etcetera; two immediately neighbouring vehicles of a platoon are adjacent. We shall mereologically model adjacency by the mereology notion of overlap.* The mereological 47 notion of proper overlap can be illustrated by **examples:** *two routes of a pipelines may overlap; and two conceptual bus timetables may overlap with some, but not all bus line entries being the same.*

2.3 Rough Sketch Hints of Domains 48

Example 5 (**Domains**) We present a number of examples:

- *Container Line:* A container line consists of a number of *container vessels* capable of holding (usually thousands of) *containers* being transported, by the vessels, between *container terminal ports* across the seven seas. A container vessel has its containers ordered in *bays*, *rows*, and *stacks* with *container terminal port cranes* depositing or removing ("lifting") *containers* onto or from port side *stack tops*. Container vessels sail specific *routes* with a route being designated by a sequence of *container terminal port visits* where a *container terminal port visit*, amongst others, has a *container terminal port name*, *estimated and actual arrival times*, etc. Etcetera. 49

- *Financial Service Industry:* A financial service industry consists of a number of *"high street" (i.e., deposit/demand) banks*, *savings & loan institutes*, *commercial banks*, other forms of banks, *insurance companies* (of differing specialisations), *stock/commodity exchanges* with their *brokers* and *traders*, one or more forms of *finance "watchdog" institutions* (SEC, FDIC, etc.), etc. A *bank* had *clients* and *clients* have one or more *accounts* having *account numbers* and *account balances* with *clients opening* and *closing accounts*, *depositing monies* into, and *withdrawing monies* from *accounts*, etc. Etcetera. 50

- *Health Care System:* A health care system consists of a number of *private physicians*, *hospitals*, *pharmacies*, *health insurance companies*, *a pharmaceutical industry*, *patients*, etc. A *hospital* consists of a number or *wards* (etc.) with each *ward* consisting of a number or *bedrooms* (etc.) with each *bedroom* consisting of a number of *beds* (etc.), etcetera. Etcetera. 51

- *Pipeline System:* A pipeline system consists of sequences of units: pumps, pipes, valves, forks and joins such that a fork connects to one pipe at the input and two at the output and a join connects two pipes at the input and one at the output, such that the first unit is a pump and is connected at the input to a well and the last unit is a valve and is connected to a sink at the output. A pump, when active (i.e., pumping) should be moving a certain volume of gas or liquid from the input to the put per time unit. A valve when closed prevents flow of gas or liquid from the input to the put, whereas when open unhindered permits such a flow. Etcetera. 52

- *Transportation System:* Transportation involves, say, three sub-domains: a transport net, a fleet of vehicles, and a community of vehicle drivers and vehicle passengers. A transport net consists of hubs and links such that a link is connected to exactly two distinct hubs and a hub is connected to zero, one or more links. Vehicles are positioned along the net: at hubs or on links and may be standing still or moving — while transporting freight, the driver and zero, one or more passengers. Etcetera.

●

In the above, rather informal, "description" of facts about specific domains we primarily focused on enumerating some of the parts. Later examples will remedy this situation.

2.4 What are Domains ? 53

So what is a domain ? We can answer this in three ways: as above, by giving examples, or, as we now do, by an informal characterisation, or by a more formal characterisation.

2.4.1 An Informal Characterisation of Domains 54

A *domain* is a set of observable entities and abstractions of these, that is, of *parts* (some of which form states), *actions* (operation applications causing state changes), *events* ("spurious" state changes not [intentionally] caused by actions) and *behaviours* (seen as set of
55 sequences of sets of actions, events and behaviours). Whereas some entities are manifested spatio-physically, that is, we can point to them, others cannot, they are either abstractions of parts, or they are actions, events and behaviours. These latter can, however, be characterised by function definitions, event predicates and behaviour definitions which [when applied] denote actions, events and behaviours.

2.4.2 A Formal Characterisation of Domains 56

A domain is a behavioural algebra described as consisting of usually two or more type descriptions, usually two or more function and event descriptions, and usually one or more behaviour descriptions, which contain channel descriptions and behaviour process descriptions.

● ● ●

Inquiry: Domain

One person's domain is another person's sub-domain (where we have yet to characterise what a sub-domain is). And: the algebra "definition" of what a domain is maybe unsatisfactory.

$\boxed{\text{MORE TO COME}}$

•

2.5 Six Examples 57

2.5.1 Air Traffic

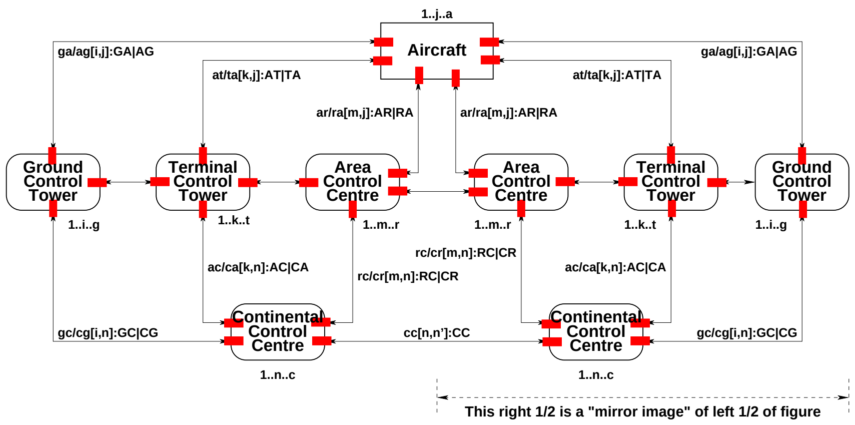

Figure 1: An air traffic system

Figure 1 shows nine (9) round edge or rectangular boxes and eighteen (18) lines. Together they form a composite part. Individually boxes and lines represent subparts. The rounded corner boxes denote buildings. The sharp corner box denote an aircraft. Lines denote radio telecommunication. Only where lines touch boxes do we have connections. These are shown as red horisontal or vertical boxes at both ends of the double-headed arrows, overlapping both the arrows and the boxes. The index ranges shown attached to, i.e., labelling each unit, shall indicate that there are a multiple of the "single" (thus representative) unit shown. Notice that the 'box' parts are fixed installations and that the double-headed arrows designate the ether where radio waves may propagate. We could, for example, assume that each such line is characterised by a combination of location and (possibly encrypted) radio communication frequency. That would allow us to consider all lines for not overlapping. And if they were overlapping, then that must have been a decision of the air traffic system.

2.5.2 Buildings 58

Figure 2 on the next page shows a building plan — as a composite part of two neighbouring, common wall-sharing buildings, A and H, probably built at different times; with room

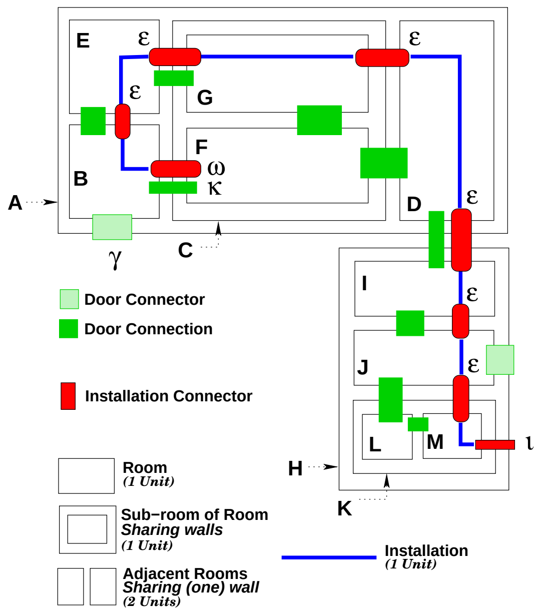

Figure 2: A building plan with installation

sections B, C, D and E contained within A, and room sections I, J and K within H; with room sections L and M within K, and F and G within C.

Connector γ provides means of a connection between A and B. Connection κ provides "access" between B and F. Connectors ι and ω enable input, respectively output adaptors (receptor, resp. outlet) for electricity (or water, or oil), connection ϵ allow electricity (or water, or oil) to be conducted through a wall. Etcetera.

2.5.3 Financial Service Industry 59

Figure 3 on the facing page shows seven (7) larger boxes [6 of which are shown by dashed lines] and twelve (12) double-arrowed lines. Where double-arrowed lines touch upon (dashed) boxes we have connections (also to inner boxes). Six (6) of the boxes, the dashed line boxes, are composite parts, five (5) of them consisting of a variable number of atomic parts; five (5) are here shown as having three atomic parts each with bullets "between" them to designate "variability". People, not shown, access the outermost (and hence the "innermost" boxes, but the latter is not shown) through connectors, shown by bullets, •.

See http://www.imm.dtu.dk/˜db/todai/tse-1.pdf

2.5.4 Machine Assemblies 60

Figure 4 on the next page shows a machine assembly. Square boxes show composite and atomic parts. Bullets, •, show connectors. Strands of two or three bullets on a thin line,

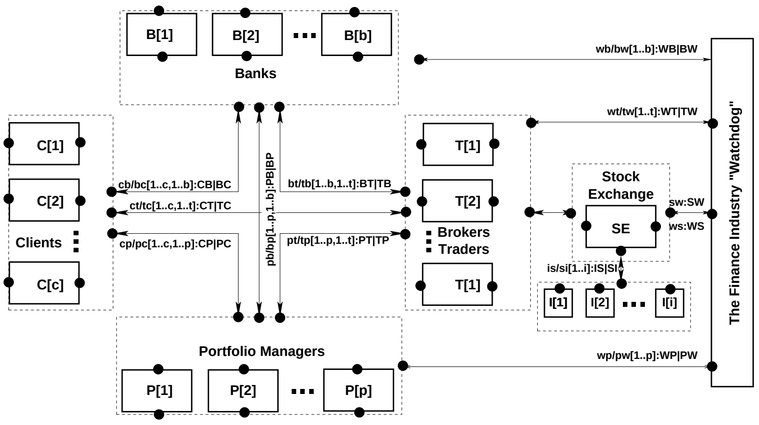

Figure 3: A financial service industry

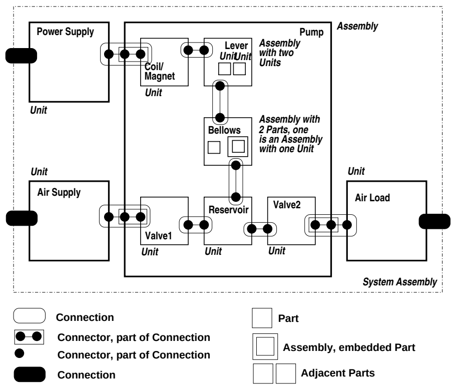

Figure 4: An air pump, i.e., a physical mechanical system

encircled by a rounded box, show connections. The full, i.e., the level 0, assembly (a composite part) consists of four parts and three internal and three external connections. The Pump is an assembly of six (6) parts, five (5) internal connections and three (3) external connectors. Etcetera. One connector and some connections afford "transmission" of electrical power. Other connections convey torque. Two connectors convey input air, respectively output air.

2.5.5 Oil Industry 61

"The" Overall Assembly Figure 5 on the following page shows a composite part consisting of fourteen (14) composite parts, left-to-right: one oil field, a crude oil pipeline system, two refineries and one, say, gasoline distribution network, two seaports, an ocean (with oil and ethanol tankers and their sea lanes), three (more) seaports, and three, say gasoline and ethanol distribution networks.

Between all of the composite parts there are connections, and from some of these

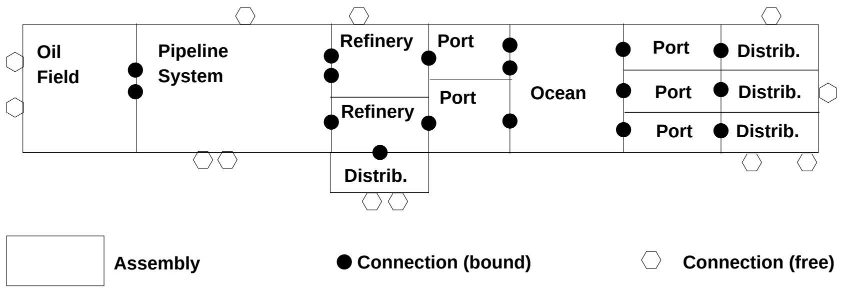

Figure 5: A Schematic of an Oil Industry

composite parts there are connectors (to an external environment). The crude oil pipeline system composite part will be concretised next.

See abstract model: http://www.imm.dtu.dk/~db/pipeline.pdf

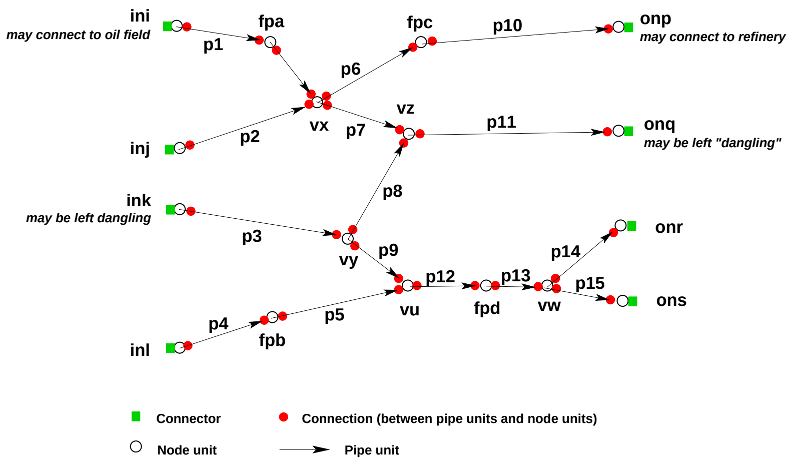

Figure 6: A Pipeline System

62 **A Concretised Composite parts:** Figure 6 shows a pipeline system. It consists of 32 atomic parts: fifteen (15) pipe units (shown as directed arrows and labelled p1–p15), four (4) input node units (shown as small circles, ∘, and labelled ini–i*n*ℓ), four (4) flow pump units (shown as small circles, ∘, and labelled fp*a*–fp*d*), five (5) valve units (shown as small circles, ∘, and labelled v*x*–v*w*), and four (4) output node units (shown as small circles, ∘, and labelled on*p*–on*s*). In this example the routes through the pipeline system start with node units and end with node units, alternates between node units and pipe units, and are connected as shown by fully filled-out **red**[9] disc connections. Input and output nodes have input, respectively output connectors, one each, and shown with **green**[10]

[9]This paper is most likely not published with colours, so **red** will be shown as **darker colour**.
[10]Shown as **lighter colour**ed connections.

2.5.6 Railway Nets

63

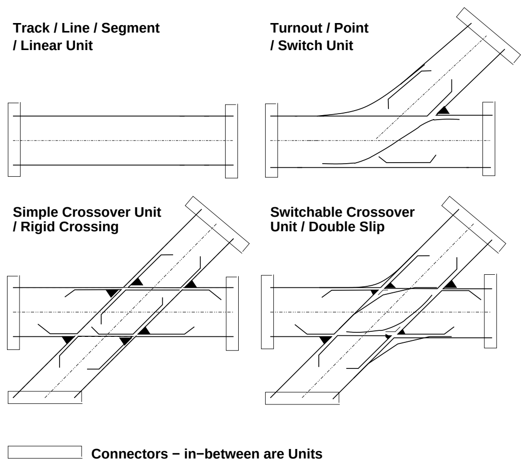

Figure 7: Four example rail units

Figure 7 diagrams four rail units, each with their two, three or four connectors. Multiple instances of these rail units can be assembled (i.e., composed) as shown on Fig. 8 into proper rail nets. 64

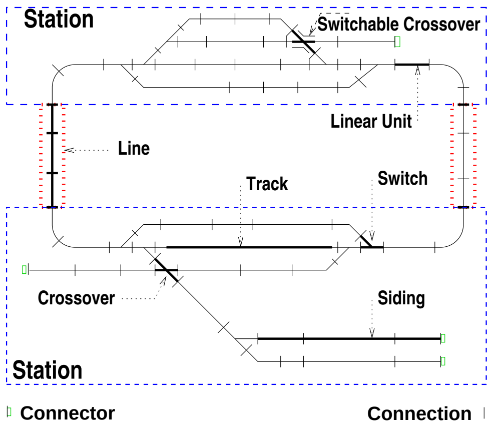

Figure 8: A "model" railway net. An Assembly of four Assemblies:
Two stations and two lines; Lines here consist of linear rail units;
stations of all the kinds of units shown in Fig. 7.
There are 66 connections and four "dangling" connectors

See http://www.railwaydomain.org/

Figure 8 diagrams an example of a proper rail net. It is assembled from the kind of units shown in Fig. 7. In Fig. 8 consider just the four dashed boxes: The dashed boxes are assembly units. Two designate stations, two designate lines (tracks) between stations. We refer to to the caption four line text of Fig. 7 on the previous page for more “statistics”. We could have chosen to show, instead, for each of the four “dangling’ connectors, a composition of a connection, a special “end block” rail unit and a connector.

3 Entities 65

By a domain entity we mean a *fact*[11] which is either an endurant entity, a *part* or is a perdurant entity, that is, an *action* an *event* or a *behaviour*. In contrast to facts we have *concepts*, that is, abstractions derived from facts. Concepts can also be considered entities. Domain entities are the things, the tangible, spatial facts we observe and the concepts we abstract from these.

Examples: 66

Example 6 (Domain Entities) One example per each of four entity categories:

- *part:* transport net;
- *action:* insertion of link;
- *event:* disappearance of a link segment (that is, fraction of a link); and
- *behaviour:* movement of vehicles along net. •

Inquiry: Fact

We say that facts are what we can observe as being a part or an action or an event or a behaviour. And we say that we can observe these.

A part, to a first "approximation", is a spatial, manifest phenomenon, something that one can point to. Is a bank a part ? Well, in "ye olde times" [still relevant Anno 2011] a bank can be presented by one or more buildings by the "books" of various kinds (for demand/deposit accounts, for mortgage accounts, for savings & loan accounts, etc.), by the cash registers, by ATMs, etc. Even though some banks may have all of the books represented electronically, they still have some physical extent: cubic spaces of electronic memory.

What part concepts may be derived from banks ? $\boxed{\text{MORE TO COME}}$

An action seems, at first, to be a concept, but it can be observed by seeing the changes of the state of physically manifested parts: change of account balance, is an example.

An event also seems, at first, to be a concept, but it can be observed by seeing the changes of the state of physically manifested parts: the disappearance of a transportation net link.

A behaviour is likewise manifest observable through its actions, events and other behaviours. •

Inquiry: Concept

So which concepts appears to be more concepts than phenomena ? An example could be a timetable. Even though there may not be any form of timetable for some (say bus) traffic, one may be allowed to say" "the busses appear to run according to some timetable". $\boxed{\text{MORE TO COME}}$ •

[11] We use the terms 'fact', 'entity', 'particular', 'thing' and 'individual' synonymously

3.1 Parts 67

By a part we understand a manifest, an endurant, that is, something we can point to, inert, possibly dynamic, i.e., animate, phenomenon or a concept thereof, something that we might (later on) represent as data by a computer.

68 **Examples:**

Example 7 (Parts) Five domain examples:

- *Container line:* container, container vessel, container terminal port, bill of lading, etc.
- *Financial service industry:* bank, bankbook, money (notes, coins), insurance policy, stock certificate, etc.
- *Transportation:* net, link, hub, vehicle, driver, etc. 69
- *Health care:* hospital, ward, bed, patient, medical staff, medical record, medicine, surgery instruments, health insurance policy, etc.
- *Pipeline system:* well, pump, pipe, valve, fork, join, sink, pipeline, etc. •

3.1.1 Atomic Parts 70

By an atomic part we shall understand a part which we, as observers, have decided form an indivisible whole, that is, one for which it is not, in a current context, relevant to speak of meaningful subparts.

71 **Examples:**

Example 8 (Atomic Parts) Five domain examples:

- *Container Line:* container, bill of lading, way bill.
- *Financial Service Industry:* bankbook, money; insurance policy; stock certificate.
- *Health Care System:* bed, patient, medical record, health insurance policy.
- *Pipeline System:* well, pump, pipe, valve, fork, join, sink.
- *Transportation System:* link, hub, vehicle, driver. •

Inquiry: Atomicity

It is the domain describer who decides, sovereignly, which parts are to be abstracted as being atomic, which parts are to be considered composite. But the domain describer does so carefully taking into consideration the scope and span of the domain description. If, for example, the domain is that of the personnel department of a company then a person may very well be considered atomic. That is, cannot be "taken apart" into head, limbs, intestinals, etc. If, instead, the domain is that of a hospital department concerned with donations of, say deceased human organs, then such a deceased may be considered composite. •

3.1.2 Composite Parts 72

By a composite part we shall understand a part which we, as observers, have decided consists of one or more proper parts also referred to as subparts.

Examples: 73

Example 9 (Composite Parts and Subparts) Five domain examples:

- *Container Line:* container vessel and its bays; bay and its rows; row and its stacks, stack and its containers.
- *Financial Service Industry:* bank and its accounts.
- *Health Care System:* hospital and its wards; ward and its bedrooms, bedroom and its beds.
- *Pipeline System:* pipeline and its wells, pumps, pipes, valves, forks, joins and sinks.
- *Transportation System:* net and its hubs and links. •

Inquiry: Composition

The remarks, above, Page 26, on atomicity, applies, inter alia, here: "It is the domain describer who decides, sovereignly, which parts are to be abstracted as being atomic, which parts are to be considered composite. But the domain describer does so carefully taking into consideration the scope and span of the domain description."

In this section, Sect. 3.1, we consider compositionality of only parts. We shall also inquire as to the compositionality of actions, events and behaviours, Sects. 3.2–3.4. •

3.1.3 Part Attributes 74

By an attribute we shall mean a pair: a type name and a value (of that type). Earlier we stated that we consider parts to be values and have types. Now we state that attributes are pairs of types and values. This must not be construed as attributes being parts. It is only that we use the concept of 'type' for two purposes: to characterise sets of parts, and to characterise individual properties of parts.

Atomic Part Attributes By the attributes of an atomic part we mean the set of properties (type names and values) that we have decided together characterise that atomic part (and all of the atomic parts of the same type).

75 **Examples:**

Example 10 (**Atomic Part Attributes**) Five domain examples:

- *Container line:* `container attributes:` length, width, height, weight, refrigerated or not refrigerated, physical location, contents, etc.
- *Financial service industry:* `account attributes:` interest rate (on loans), yield (on deposits), owner(s), maximum credit, current balance, etc.
- *Health care:* `patient attributes`: name, central personal registration identifier, gender, birth date, birth place, nationality, weight, height, insurance policies, medical record, etc.
- *Pipeline system:* `pipe attributes`: circular diameter, length, location, maximum laminar flow, current flow, guaranteed maximum leak (in volume/second), current leak, etc.
- *Transportation:* `link attributes`: length, location, link state (open in one direction or the other or open in both or closed in both directions), link type (road, rail, sea, air), etc. •

76

Composite Part Attributes By the attributes of a composite part we mean the set of properties (type names and values) (exclusive of all subparts of that composite part) that we have decided together characterise that composite part (and all of the composite parts of the same type).

77 **Examples:**

Example 11 (**Composite Part Attributes**) Five domain examples:

- *Container Line Attributes:* name of container line, for example MAERSK, legal residence (address), incorporated ?, responsible capital, organisational structure (explicated organigram), subsidiaries, budget, accounts, etcetera.
- *Financial Service Industry Attributes, Bank:* name of bank, kind of bank [whether a demand/deposit or a savings & loan or an investment bank or other], legal residence (address), responsible capital, organisational structure (explicated organigram), subsidiaries, budget, accounts, etcetera. 78

- *Health Care System Attributes, Hospital:* name, kind of hospital [whether a general hospital or a specialised hospital, and then its speciality], legal residence (address), legal owner, organisational structure (explicated organigram), sources of financing, budget, accounts, etcetera.
- *Pipeline System Attributes:* name of pipeline system[12], legal residence (address), legal owner, sources of financing, geography, maintenance subcontractors, budget, accounts, etcetera. 79
- *Transportation System Attributes:* name of transport system, kind of transport system[13], legal residence (address), legal owner, sources of financing, geography, maintenance subcontractors, budget, accounts, etcetera. •

• • •

Inquiry: Attribute and Property

We shall use the concept of 'property' in a wider sense than that of 'attribute'. This broader concept of 'property' has been studied by philosophers [18, 20, 34]. MORE TO COME
•

80

Static Part Attributes A part attribute is static if that part never changes its value.

Examples

Example 12 (Static Part Attributes) Two examples:

- **Patients:** name, central personal registration identifier, gender, and birthplace.
- **Links:** length. •

81

Dynamic Part Attributes A part attribute is dynamic if that part can change its value.

Examples

Example 13 (Dynamic Part Attributes) Three examples:

- The height, weight, blood pressure, blood sugar, temperature, and (hence) patient medical record of a patient are dynamic attributes. 82
- A hub typically can connect a number of distinct links and thus can attain either one of number of hub states each hub state being a possibly empty set of pairs, $(\mathsf{li}_j,\mathsf{li}_k)$, of not necessarily distinct link identifiers (li) of the links connected to that hub.

[12]for example: Nabucco http://en.wikipedia.org/wiki/Nabucco_pipeline
[13]whether a road system or a rail/train system, or an airline, or a shipping company, etc.

The state of a hub is a dynamic attribute.

- Similarly for link states. •

83

Indivisibility of Attributes Given a part of some kind (i.e., having some set of attributes), whether atomic or composite, one cannot "remove" an attribute from that entity and still retain the entity as being of that kind.

Examples

Example 14 (Indivisibility of Attributes) Two examples:

- One cannot remove the attribute 'height' from an entity of kind person
- and one cannot remove the attribute 'kind of transport system' from an entity of kind 'transport system'. •

Inquiry: Atomic Parts and Highly Structured Attributes

Let us analyse an example domain: that of banking. A bank has clients; clients have accounts; a clients may have zero, one or more accounts; two or more clients may share accounts (i.e., multiple accounts); and accounts register which transactions (`open, deposit, withdraw, get_statements, transfer, close`) have been performed on the account, its balance, its interest rate to be paid by clients if account balance is negative, its yield accrued to the client when the account balance is positive. Accounts have account numbers. Etcetera. The terms written in the sans serif font designate parts (and, as we shall see later, also behaviour — distinct from parts). The terms written in slanted, italic font denote attributes. The terms written in `teletype font` designate actions. Now to an analysis. In a financial service system there are many clients (seen as behaviours) with contexts and states (also named clients), and many banks (likewise seen as behaviours etcetera). The fact that a bank may have several branch offices must not be confused: these branch offices are not separate banks: they all share the same clients and the same accounts. We shall see a bank as a concept and as a set of one or more branch offices (including the one located at the head quarters of the bank). The bank concept is a part and the branch offices are a set of parts. The bank concept has a number of attributes: customers ("proxies" for, but not to be confused with, clients), accounts, account sharing, multiple customer accounts, etc., with accounts having sub-attributes: owners, balance, interest, yield, etcetera. The bank concept part is shared by all branch offices. The branch offices are usually physically embodied and distinct. The bank concept ism in principle, not physically embodied: one cannot point to it. It may be "implemented" physically in terms of "ye olde" ledgers (books), or in terms of a large, central database or in terms of a set of smaller, distributed databases, say one per branch office, or in terms of client-held smart cards, etcetera. The branch offices may be hard to identify physically: they could be physical buildings and offices, or they could be ATMs: automatic teller machines, or they could be the client-held smart cards. The essence of the above is that a bank is an atomic entity with highly structured attributes. A model of these attributes could be:

1. *A bank has as attributes*
   - a *a name,*
   - b *a set of uniquely identified customers,*
   - c *a set of uniquely identified accounts,*
   - d *a "book" which records the accounts of each customer,*
   - e *a "book" which records the owners of each account.*
   - f *etcetera.*
2. *An account has*
   - a *a balance,*
   - b *a chronologically identified set of transactions hitherto performed on the account,*
   - c *an interest on negative balances,*
   - d *a yield on positive balances,*
   - e *etcetera,*

```
type
1.   Bank ::
1a.       BankName
1b.       CustId-set
1c.       AccId →m Account
1d.       CustId →m AccId-set
1e.       AccId →m CustId-set
2.   Account ::
2a.       Balance
2b.       DateTime →m Transaction
2c.       Interest
2d.       Yield
```

Truly a highly structured set of attributes and sub-attributes. •

Inquiry: Parts, Behaviours (Agents) and Attributes

There is a commonly misunderstood dichotomy[14]*: parts as possibly dynamic, inert entities, and behaviours. Take the term 'train'. We may speak of "the train" as a composite (or even as an atomic) part — such as it is manifested on the train station platform at some time; we may speak of "the train" as a behaviour such as it is manifested when it speeds down the rails. We may even extend this dichotomy beyond two mutually exclusive or contradictory entities to also include part attributes. Thus we may speak of "the train" as an attribute — such as it is manifested by a specific entry in a timetable, or as in "she took the 4:50 from Paddington". Of course, the confusion arises from our use of the same term 'train' in all these cases. (We invite the reader to formulate appropriately distinct 'train terms'.) But the message should be clear: That is describing the train behaviour one needs refer to the train part(s). Thus the catchphrase: to some a "thing" is a part, to others it is a behaviour.* •

[14]Dichotomy: a division into two mutually exclusive or contradictory entities.

3.1.4 Subparts Are Parts 84

By a subpart, p', of a part, p, we thus mean an entity which is not the same as the part, that is $p \neq p'$. We say that a part, p', is a *proper part* of another part if it is a subpart of that part. So by proper part of p and subpart of p we mean the same.

Examples

Example 15 (Sets of Hubs and Hubs – Sets of Links and Links) From a net we observe sets of hubs and sets of links: A set of hubs is a value of the type sets of hubs. A hub is a value of type hub. A set of links is a value of the type sets of links. A link is a value of type link. •

3.1.5 Subpart Types Are Not Subtypes 85

Thus, by a subpart type we mean a part type but the type of the subpart cannot be the same as the type of the part of which it is a subpart.

Examples

Example 16 (Part and Subpart Types) We refer to Example 15. Let a part be a transportation net, *n:N*. A subpart of a transportation net, *n:N*, is, for example, the part *hs:HS*, which is the set of all hubs of the net, and a hub, *h:H*, which is a part of *hs:HS*, is a subpart of *hs:HS*. And all these subparts are of different types, to wit: *HS* and *H*, and, as we shall see, *LS* and *L*, are not subtypes of type *N*. •

86 By a 'union', *A*, of *disjoint types*, say *B*, *C*, ..., *D*, that is: $A{=}B|C|...|D$, we mean a type whose values are either of type *B* or of type *C* or ... of type *D*, and where every type value is of exactly one of the types *B*, *C*, ..., *D*. These types, *B*, *C*, ..., *D*, are subtypes of *A*.

Thus subpart types are not the same as subtypes of the part of which the subpart is a proper part.

To be consistent we rule out the possibility of defining types recursively.

3.1.6 Mereology of Composite Parts 87

By the mereology of a composite part we understand the number of subparts of respective kinds (types) of that composite entity and how the subparts are related to one another.

88

Examples

Example 17 (Mereology of Composite parts) Five domain examples:

- *Container Line System:* A container vessel contains a number of uniquely identified bays, bays consists of a sequentially indexed sequence of (usually several) rows, and rows consist of a sequentially indexed sequence of (usually several) stacks, and stacks consists of a sequence of zero or more containers — such that access to stacks are by identity of bay, number of row, number of stack and then to the top of this possibly empty stack. Etcetera.

- *Financial Service Industry:* A bank consists of (i) a set of uniquely identified demand/deposit accounts, (ii) a set of uniquely identified savings & loan accounts, (iii) a set of uniquely identified mortgage accounts. Etcetera. 89
- *Health Care System:* A hospital consists of (1) a set of uniquely identified wards of kind κ_1, (2) a set of uniquely identified wards of kind κ_2, ..., and (n) a set of uniquely identified wards of kind κ_n. Etcetera.
- *Pipeline System:* A pipeline system consists of a set of units — where units a either wells, pumps, pipes, valves, forks, joins or sinks — and such that (a) a well is connected to one or more pumps, (b) a pipe is input-connected to either a pipe or a pump or a valve or a fork or a join and is output-connected to either a pipe or a pump or a valve or a fork, (c) a pump is input-connected to a pipe and is output-connected to a pipe, (d) a valve is input-connected to a pipe and is output-connected a pipe or a sink, (e) a fork is input-connected to a pipe and is output-connected to two pipes, (f) a join is input-connected to two pipes and is output-connected to a pipe, and (g) a sink is input-connected to a valve. Etcetera. 90
- *Transportation System:* A transport net consists of a set of hubs and a set of one or more links such that links connect exactly two distinct hubs, and thus such that hubs are connected to zero or more distinct links. Etcetera.
  - The mereology of a net can be expressed in terms of unique identifiers associated with hubs, hij, hik, . . . , him, and links, lia, lib, . . . , lic. •

91

Mereologically two parts, e_i, e_j, may stand in the following relationships: (a) either e_i is identical to e_j, (b) or e_i is fully disjoint from e_j, (c) or e_i is adjacent (i.e., connects) to (disjoint from, but "touches") e_j, (d) or e_i is fully contained within e_j, (e) or e_i partially overlaps with e_j (that is, there are "areas" of e_i which are not overlapping with "areas" of e_j).

3.1.7 Part Descriptions 92

To describe an atomic part (type) it suffices to describe all the atomic part attributes: its type name, the attributes, and its possible contribution to the mereology of "a whole": own unique identification, and how it 'unique identifier'-relates to other parts.

To describe a composite part (type) it is necessary to describe these things: (i) all the composite part attributes, (ii) each of the subpart types (i.e., subparts), and (iii) their mereology.

93

Examples

Example 18 (Description of An Atomic Part) We continue our example of transport nets. A link is here considered an atomic part.

Type Name: link.

Attributes: length, location[15], current state[16] and state space[17], etc.

Unique Identification: unique Link identifier.

Mereology: a pair of unique hub identifiers. •

94

Example 19 (Description of A Composite Part) We continue our example of transport nets. A net is here considered a composite part.

Type Name: net.

Attributes: name, transport kind, legal address, legal owner, sources of financing, geographical area, maintenance subcontractors, budget, accounts, etc.

Unique Identification: `not applicable`.

Mereology: `not applicable`.

Subpart Type[s]: set of links, set of hubs. •

3.1.8 States 95

By a state we understand a specific set of parts such that for each of these parts some attributes are dynamic.

96

Examples

Example 20 (States) Five domain examples:

- *Container line:* container, container vessel, container terminal port.
- *Financial service industry:* bank (as a whole), account (as a subpart).
- *Health care:* hospital, ward, bed, patient.
- *Pipeline system:* well, pump, pipe, valve, pipeline.
- *Transportation:* net, link (open in one direction, open in the opposite direction, open in both directions; closed in all [two] directions), hub (open between a specific [possibly empty] set of pairs of links connected to the hub), vehicle. •

[15]The cartographic and cadestral location of a link may, amongst other components, include, for example, a Beziér curve description of how that link "traverses" a, or the landscape.

[16]in terms of sets of pairs of distinct identifiers of connecting hubs

[17]in terms of sets of possible link states

Inquiry: State

> *What is a state and what is not is sometimes an elusive issue. If the outcome of each of a set of operations on parts is independent, say over long time intervals, say years for a transport system, then it seems that the arguments of these operations do not contribute to a state notion. If, however, the time span being considered is such that the outcome of operations being carried out depend very much on argument values, then these do indeed contribute to a state notion. So the notion of a state has to do with whether part values are constant over very long periods or whether they vary quite often. Please note that we only very roughly referred to a notion of time interval without being specific.* •

Inquiry: Environment

> *In 'inquiry'* `State`*, above, we alluded to some operations being dependent as to their varying outcome on a state. In contrast these or other operations may require arguments whose value remain constant over "large" time intervals. We say that these arguments for an environment. Other than this mentioning we shall not deal with the notion of environment in these notes.* •

3.2 Actions 97

By an action we understand a state change resulting directly from the expected application of a specific function (one of several possible), that is, the specific function was performed deliberately, on purpose.

98

Examples

Example 21 (Actions) We give examples from five domains. The examples are not proper descriptions of actions. We basically just give their names. These names — and the familiarity of the domains — are such that the reader is "tricked into" thinking: *"oh yes, I see; but, of course."* Only a proper action description can reveal the action. 99

- *Container line:*
  - loading a container;
  - unloading a container;
  - moving a container from one location (say on-board a vessel) to another location (say in a container terminal port).
- *Financial service industry:*
  - open an account,
  - deposit money into an account,
  - withdraw money from an account,
  - obtain account statement,
  - close account.

100

- *Health care:*
  - admitting a person as a patient;
  - allocating a bed in a ward to a patient;
  - medicating a patient.
- *Pipeline system:*
  - opening a pump (for pumping);
  - closing a valve.
- *Transport Net:*
  - inserting a hub;
  - inserting a link;
  - removing a hub;
  - removing a link. •

Inquiry: Action, Operation, Function

We shall, perhaps somewhat arbitrarily, be making a distinction between the concepts of function, action and operation.

By a function we shall understand something which when applied to something called its arguments yield something called the result of that function for those arguments.

By invocation we mean the same as application.

By an action we shall mean a function application which potentially changes a state.

By an operation we shall understand a function application which is like an action, or does not change such a state. •

3.3 Events

101

By an event we understand a state change resulting indirectly from the unexpected application of a function, that is, the specific function was performed "surreptitiously", Events can be characterised by a pair of (before and after) states, a predicate over these and a time.

Events are thus like actions: change states, but are usually either caused by "previous" actions, or caused by "an outside action".

102

Example

Example 22 (Events) Five domain examples:

- *Container line:* A container falls overboard.

- *Financial service industry:* A bank goes bankrupt.
- *Health care:* A patient dies.
- *Pipeline system:* A pipe breaks.
- *Transportation:* A link disappears. •

Inquiry: Event

The characterisation of 'event' given above is far from satisfactory. The event concept characterisation is pragmatic. A more satisfactory characterisation might be:

> *An event can be described by a predicate, by a time (point), and a pair of states (the "before the event" and the "after the event") such that the predicate holds for these two states [and the time].*

In these notes we do not ascribe time points with the occurrences of actions. That should be done in subsequent work. Likewise we do not ascribe time points with the occurrences of events.

The philosophic concept of event is treated by, for example, [15, 39, 33, 24, 38, 9, 14].

•

3.4 Behaviours 103

By a behaviour we understand a set of sequences of actions, events and behaviours.

104

Example

Example 23 (Behaviours) Five domain examples:

- *Container line:* The transport of a container from it being fetched at the sender, via a sequence of one or more triplets of loadings onto a vessel, unloading at another container terminal port and possibly temporary storage at that port, to its final delivery at a receiver. 105
- *Financial service industry,* account handling: the opening of an account, a sequence of deposits, withdrawals and statements to the closing of that account. 106
- *Health care,* patient hospitalisation: the admission of a patient to a hospital, initial anamnese, analysis, diagnostics and treatment plan, via an alternating sequence of treatments (including surgical operations), repeated analyses, evaluations and possible reformulation of diagnostics and treatment plan, to a final discharge. 107
- *Pipeline system,* simple, day-to-day operations. The flow of gas (or a liquid) through a pipeline net: pumped from wells, fed through pipes, valves, forks and joins, to leaving the net at sinks. 108

- *Transportation:* The movement of a vehicle along a transport net:

  from positions at hub or link positions via a sequence of zero, one or more hub and link movements, to a final hub or link position.

  Example 4 (Pages 10–14) illustrated a transport behaviour. •

Inquiry: Behaviour

The notion of behaviour is not the same as the notion of process. We shall reserve the use of the term behaviour for "what goes on in the domain", and we shall use the term process for "what goes on 'inside' the computer". •

3.5 Discussion 109

We have dealt, in some detail, with the concept of parts (Sect. 3.1, Pages 26–35). Our "corresponding" treatment of actions, events and behaviours (Sects. 3.2–3.4, Pages 35–38) have been far less detailed. The reason for this is the following. Types emerge (Sect. 3.1) as a means of describing parts. And types are indispensable in the description of action, event and behaviour signatures (Sects. 3.2–3.4). Types thus form the very basis for the description of all entities. And we have chosen to let the type concept emerge from our 110 treatment of parts. There is another reason for Sect. 3.1 being somewhat more detailed than Sects. 3.2–3.4. When studying parts we could, relatively easily, introduce such notions as atomic and composite parts, attributes of these, and mereologies of composite parts. These notions, under some disguise, can likewise be found for actions, events and behaviours, but they are not that easily introduced.

4 Describing Domain Entities 111

4.1 On Describing

The purpose of description is to use for example informal text to present an entity (simple, action, event or behaviour) so that the reader may "picture" ("envisage"), that which is being described. The text describing the entity is said to be a syntactic quantity. and the entity is then said to be a semantic quantity: the syntactic text *denotes* the semantic quantity. We also say that the syntactic quantity designates, denotes, indicates, specifies, 112
points out, gives a name or title to, or characterises[18] the semantic quantity.

4.1.1 Informal Descriptions

In the many examples[19] of Sects. 2–3 we have made several references to quite a few domain entities. We do not claim that we have described these entities.

113
Domain Instances Versus Domains What we can observe are instances of a specific domain or fragments (perhaps parts) of a specific domain. What we describe are either abstractions of these instances or abstractions of a set (i.e., a type) of these instance. If 114
someone describes me as an atomic part with the action(s) and event(s) of my behaviour, then that someone describes an instance of a person, not the domain of all persons, but in that description it is expected that *many fragments* of the description is also valid for either a lot of persons or all persons. We say that these *many fragments* describe not an instances but fragments of abstractions of a domain of persons.

115
Non-uniqueness of Domain Descriptions We say 'a domain', not 'the domain'. Two or more domain describers may not exactly focus on the same entities and their properties. A domain description is always an abstraction. Something is left out. Not all entities and not all properties of those entities included may be deemed worthwhile to be included.

A good domain description, to us, is a domain description that covers what most stake holders can agree on to be relevamt aspects of the domain, that reveals generally unknown facets of the domain, and that is terse and precise.

116
A Criterion for Description For us, to informally describe an entity ideally means the following: *Let there be given what we can agree on to be an entity, call it e. Let there be given what is claimed to be a description of that entity. Let a person read and claim to have understood that description. Now that person is confronted with some phenomenon e'. Either that phenomenon is the same or it is of the same kind (type) as e or it is not. If e' is of the same kind as e then the person must identify it as such, unequivocally. If e' is not of the same kind as e then the person must identify it as not being so, likewise unequivocally.* 117

[18]— eight alternative terms for the same idea!
[19]Examples 5–23

If a description does not satisfy the above then it is not a *proper description*.

The above "criterion" suffers, seriously, from our not having made precise what we mean by *"same"* and *"same kind"*.

These notes are not the place for a much needed investigation of the "sameness" problem. It is basically a philosophical question. But we should not overlook the fact that it is the domain describer and the domain stake holders who, finally, decide on "sameness".

118

Reason for 'Description' Failure There can be three reasons for a description to not be proper:

1. *either all phenomena are entities as described — the description is vacuous;*
2. *or there are entities which were meant to be of the type or not meant to be of the type described but which "fall outside", respectively "fall inside" the description;*
3. *or the description does not make sense, is "gibberish", ambiguous, or otherwise.*

That is: a proper description, when applied to entities, "divides" their world into two non-empty and disjoint sets: the set of all entities being described by the description, and the rest !

119

Failure of Description Language But we have a problem ! One cannot give a precise definition of exactly the *denoting language*, that is, of exactly, all and only those informal texts which designate entities. Firstly, we have not given a sufficiently precise informal text characterisation of entities, Secondly, natural (cum national) languages, like English, defy such characterisations. We must do our best with informal language descriptions.

120

Guidance But there is help to be gotten! The whole purpose of Sect.3 was to establish the pointers, i.e., guidelines, as to what must be described, generally: parts, actions, events and behaviours, and specifically: whether atomic or composite parts, their attributes, and, optionally, their mereology, and, for composite parts, their subparts; and, as a starter, the signatures of actions, events and behaviours. This section will continue the line reviewed just above and provide further hints, pointers, guidelines.

4.1.2 Formal Descriptions 121

We shall, in addition to the *description components*[20], outlined in Sect. 3 now join the possibility of improved description precision through the use of formal description. We argue that formal description, while being used in-separately with precise informal narrative. improves precision while enabling formal proofs of properties of that which is denoted by the description. 122

[20]parts, actions, events and behaviours; attributes and possibly unique identifiers of parts, and mereology of composite (atomic) parts; subparts of composite parts; etc.

We shall here use the term 'formal' in the sense of mathematics. A formal description language is here defined to have a formal syntax, that is, a set of syntax rules which define precisely and unambiguously, which texts over the alphabet of the language are indeed sentences of that language[21], a formal semantics, , that is, something which to every syntactically valid sentence of the language, ascribes a meaning in terms of a mathematical quantity[22], and a proof system, that is, a consistent and relative complete set of axioms and proof rules using which one can prove properties of descriptions.

We shall "unravel" an example formal description language, `FDL`, in this section. `FDL` has similarities to the `RAISE` [23] Specification Language, `RSL` [22], but, as our informal explanation of the meaning of `FDL` will show, it is not `RSL`. The similarities are "purely" syntactical.

4.2 A Formal Description Language 123

4.2.1 Observing and Describing Entities

We make the obvious distinction between observing semantic values but expressing syntactic structures. We observe parts, but first express types and then their properties; actions, but first express their signatures and then their definitions; events, but first express their signatures and then their definitions; and behaviours, but first express their signatures and then their definitions.

4.2.2 Observing and Describing Parts 124

In order to describe a part we use such phrases as: *a patient* (whom we here consider to be an atomic part)) is characterised by as set of properties a `name.` a `central personal registration identifier,` a `gender,` a `birth date,` a `birth place,` a `nationality,` a `weight,` a `height,` a `insurance policy,` a `medical record,` etcetera; and *a transport net* (whom we here consider to be a composite part)) is characterised by a set of properties a structure of, in this case two subparts, ie., a set of hubs and a set of links, and their mereology. Thus we take the nouns *name, central personal registration identifier, gender, birth date, birth place, nationality, weight, height, insurance policy, medical record, . . . , set of hubs,* and *set of links* as type names. The names 'patient' and 'transport net' are also domain names. That is, we go from instance of part to the type of all parts "of the same kind". One must take great care in not confusing the two: type and value). Later we shall clarify the distinction between type and domain names. 125 126

127

Abstract Types By an *abstract type* we generally mean some further unexplained set of mathematical quantities. Abstract types are in contrast to concrete types by which

[21]that is, the alphabet and sentences can be considered mathematical quantities

[22]a set, a Cartesian, a list, a function, or some such mathematical item which can be characterised by a number of properties

we mean such mathematical quantities as sets, Cartesians, lists, maps and functions (in general). Abstract types are also referred to as *sorts*.

The FDL clause:

type A

defines what we shall here, simplifying, take as a set of values said to be of type A. A is said to be a *type name* (here, more specifically, a *sort name*).

128

Concrete Types Borrowing from RSL, and, in general, discrete mathematics, we introduce FDL clauses for expressing set, Cartesian, list, map and function types. Let A, B, ..., C be (type or sort) names which denote some (not necessarily distinct) types, then

A-**set**, (A $\times$ B $\times$ ... $\times$ C), A*, A$^\omega$, A $\overrightarrow{m}$B, A$\rightarrow$B, A$\overset{\sim}{\rightarrow}$B

are *type expressions* which denote the following (left-to-right) concrete types: set of sets of type A values; sets of Cartesian values, (a,b,...,c), over types A, B, ..., C; set of finite lists of elements of type A values; set of possibly infinite lists of elements of type A values; set of maps, that is enumerable functions from type A into type B values; set of total functions from type A into type B values; respectively set of partial functions from type A into type B values. 129 Choosing to describe a part as a sort rather than a concrete type reflects a *principle* of abstraction. Modelling a concrete type in terms of, for example, a map type (A $\overrightarrow{m}$B) rather than as type of indexed sets ((A$\times$B)-**set**) reflects a modelling *technique*.

130

Type Definitions Besides the sort type definitions, e.g., **type** A there are the concrete type definitions.

Let D be some (unused)type name, then

type D = Type_Expression

is a concrete type definition where Type_Expression is of either of the forms A-**set**, A-**infset**, (A$\times$B$\times$...$\times$C), A*, A$^\omega$, A $\overrightarrow{m}$B, A$\rightarrow$B, A$\overset{\sim}{\rightarrow}$B and A|B|...|C, where A, B, ..., C are either type names or, more generally, other such type expressions and where A|B|...|C expresses the "union" type of the A. B. ..., and C types. 131

Example 24 (Transport Net Types) Let us exemplify the above by starting a series of examples all focused on a domain of transport nets.

132 Figure 9 on the facing page shows a net with eight hubs and seven links.

To be able — here, in this tect — to refer to fragments (here sub-parts), of what is shown in Fig. 9, we label the parts with names (Fig. 10); these names stand for the designated parts. They are not properties of the parts, they are the parts. Also: they are not the unique indentifiers of the parts. 133

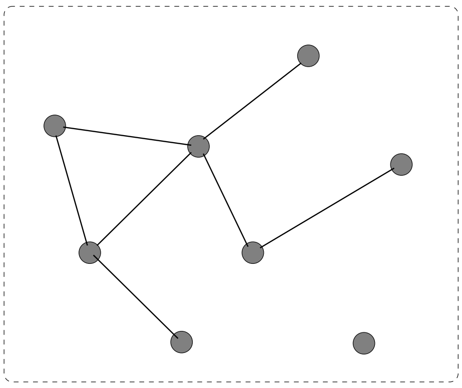

Figure 9: A transport net

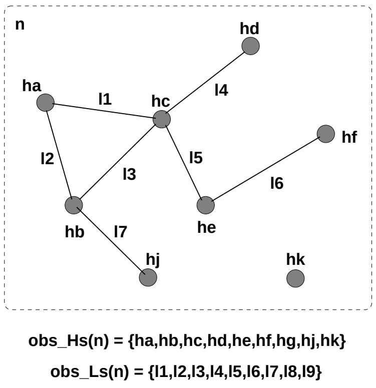
n
hd
ha
l1
hc
l4
hf
l2
l3
l5
l6
hb
l7
he
hj
hk
obs_Hs(n) = {ha,hb,hc,hd,he,hf,hg,hj,hk}
obs_Ls(n) = {l1,l2,l3,l4,l5,l6,l7,l8,l9}

Figure 10: A transport net

32. We focus on the transport net domain. That domain is "dominated" by the composite parts of nets, *n*:*N*.

32. **type** N

33. There are two subparts of nets:

    a sets, hs:HS, of hubs (seen as one part) and

    b sets, ls:LS, of links (also seen as one, but another part).

33a. HS

33b. LS

As part of identifying the composite net type, N, we also identify two observers: obs_HS (observe [set of] hubs) and obs_LS (observe [set of] links) That is: 134

33a. obs_HS: N → HS 33b. obs_LS: N → LS

34. Hubs subparts of HS ('sets of hubs'), and
35. links are subparts of respectively LS ('sets of links'),

    and are of types

    a H, respectively
    b L.

type
35a. H
35b. L

135 We may, for convenience, bypassing a step (i.e., Items 33a–33b) instead express:

type	**value**
35a. Hs = H-**set**	35a. obs_Hs: N → H-**set**
35b. Ls = L-**set**	35b. obs_Ls: N → L-**set**

•

136 **Type Properties** In the following we shall be introducing a number of functions which analyse parts with respect to respective *properties* [18, 35, 21]. There are three kinds of properties of interest to us: the subparts of composite parts, the mereology of composite parts, and general attributes of parts (apart from their possible subparts and mereology). 137 Every entity, whether simple, or an action, or an event, or a behaviour, has a unique identification. The mere existence, in time and space, endows a part with a unique identification as follows. No two spatial parts can occupy overlapping space, so some abstract spatial location is a form of unique identification. We consider the unique identity of a part of type A, say AI, as a general attribute. We use the attribute values of AI to formulate 138 mereologies.

Thus there are three kinds of property analysis functions.

- *Subpart observer functions*, obs_B, obs_C, . . . , obs_D, which apply to composite parts (say of type A), and yield their constituent subparts, say of type B, C, . . . , D:

  obs_B: A → B, obs_C: A → C, ..., obs_D: A → D;

- *Mereology functions* which apply to composite parts (say of type A), and yields elements of their mereologies.

  Let parts of type B, C, ..., D be in some mereology relation to parts of type A. That is, there are mereology functions mereo_Ax: B → AIx, mereo_Ay: C → AIy, ..., mereo_Az: D → AIz

where Ax, Ay and Az are some distinct identifiers and AIx, AIy and Ayz are some type expressions over type A, typically

AI, AI-**set**, (AI×...×AI), etc.

139

- *Attribute Functions* which apply to parts (say of type A) and yield their attributes (short of the mereologies) (say of types E, F, ..., G:

  attr_E: A → E, attr_F: A → F, ..., attr_G: A → G;

  Among the general attribute functions are the unique identification functions, say attr_A.

140

Subpart Type Observers Given a composite sort, named, say, A, and postulating that its values contains subparts of type B, one can observe these type B subparts of A using the likewise postulated *observer function*:

obs_B: A → B

If A values also contain subparts of types C, . . . , E, then there also exists the additional observer functions: obs_C, . . . , obs_E. We say that the observer functions are postulated. We postulate them. And we endow them with properties so that they "stand out" from one another. First examples of properties are given by the observer function signatures: from type A values observer function obs_B yields B values. Further properties may be expressed through axioms. 141

Example 25 (Subpart Type Observers) From nets we observe

36. sets of hubs and
37. sets of links.

in either of two ways:

value	**value**
36. obs_HS: N → HS	35a. obs_Hs: N → H-**set**
37. obs_LS: N → LS	35b. obs_Ls: N → L-**set**

•

142

Unique Identifier Functions All parts have unique identifiers. This is a dogma. We may never need some (or any) of these unique part identifiers. But they are there nevertheless.

Example 26 (Unique Hub and Link Identifiers) From hubs and links we observe their unique

38. hub and

39. link

identifier attributes and their 'observers':

type
38. HI
39. LI

value
38. uid_HI: H → HI
39. uid_LI: L → LI

143

Figure 11 shows the labelling of links with unique link identifiers, and of hubs with unique hub identifiers. It also shows sample unique identifier observer functions.

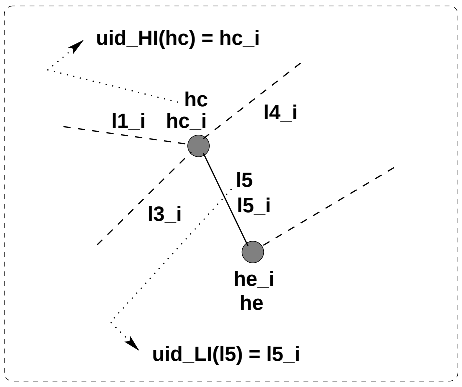

Figure 11: Fragment of a transport net emphasizing unique part identfiers and their observers

•

144

Mereologies and Their Functions

Example 27 (Transport Net Mereology) To express the mereology of transport nets we build on the unique identifications of hubs and links.

40. Links connect exactly two distinct hubs, mereo_HIs.

41. Hubs are connected to zero, one or more distinct links, mereo_LIs.

type
40. HIs = HI-**set**
axiom
40. ∀ his:HIs•**card** his=2
type
41. LIs = LI-**set**
value
40. mereo_L: L → HIs
41. mereo_H: H → LIs

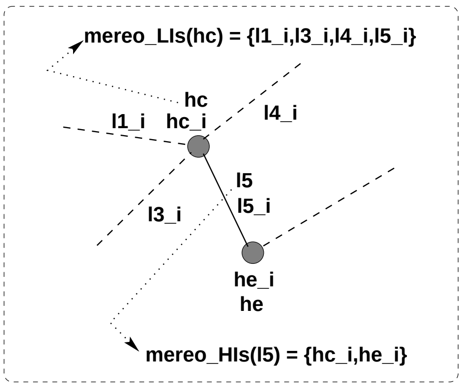
mereo_LIs(hc) = {l1_i,l3_i,l4_i,l5_i}
hc
l1_i
hc_i
l4_i
l5
l5_i
l3_i
he_i
he
mereo_HIs(l5) = {hc_i,he_i}

Figure 12: Fragment of a transport net emphasizing mereology observers

145 146 Figure 12 illustrates the idea of mereology observer. The above (Items 40–41) form the basis for expressing the constraints on how hubs and links are connected.

42. Given a net, its link and hub observers and the derived link and hub identifier extraction functions, the mereology of all nets must satisfy the following:

    a All link identifiers observed from hubs must be of links of that net and

    b All hub identifiers observed from links must be of hubs of that net.

43. We introduce two auxiliary functions for extracting all hub and link identifiers of a net.

147

axiom
42. ∀ n:N,
42. **let** ls=obs_LS(n),hs=obs_HS(n),
42. lis=xtr_LIs(n),his=xtr_HIs(n) **in**
42a. ∀ h:H•h ∈ hs ⇒ mereo_H(h)⊆lis ∧
42b. ∀ l:L•l ∈ ls ⇒ mereo_L(l)⊆his **end**
value
43. xtr_LIs: N → LI-**set**, xtr_HIs: N → HI-**set**
43. xtr_LIs(n)≡{uid_LI(l)|l:L•l ∈ obs_LS(n)}
43. xtr_HIs(n)≡{uid_HI(h)|h:H•h ∈ obs_HS(n)}

•

148

General Attributes and Their Functions

Example 28 (**Hub States and Hub State Spaces**) In addition to the unique identifiers and the mereology of parts there are the general attributes. An example are the states of hubs and links where these states indicate the direction of traffic for which the hubs and links are open. 149

44. With any hub, h, we thus associate
    - a a hub state, $h\sigma$, consisting of a set of pairs of link identifiers (with (li_j, li_k) in $h\sigma$ expressing that traffic is open from link l_j to link l_k via hub h), and
    - b a hub state space, $h\omega$, consisting of a set of hub states.
45. The relations between
    - a the link identifiers of the hub,
    - b the hub states, and
    - c the hub state spaces
46. must satisfy the following
    - a wrt. the potential set, hs, which is the "largest possible" hub state for h, one that allows traffic from any link l_i incident upon h to any link l_j emanent from h:
    - b the hub state is any subset of hs, and
    - c and such hub state is in that hub's state space.

150

```
type
44a.  HΣ = (LI×LI)-set
44b.  HΩ = HΣ-set
value
44a.  attr_HΣ: H → HΣ
44b.  attr_HΩ: H → HΩ
axiom
45.   ∀ n:N,h:H • h ∈ obs_Hs(n) ⇒
41.       let hlis = mereo_H(h),
44a.          hσ = attr_HΣ(h),
44b.          hω = attr_HΩ(h),
43.           lis = xtr_LIs(n) in
46a.      let hs = {(li,lj),(lk,li)|li:LI•li ∈ hlis∧{lj,lk}⊆lis} in
46b.      hσ⊆hs  ∧
46c.      hσ ∈ hω end end
```

●

151

Example 29 (Further Atomic Attributes) In addition to the unique link identifier links also have, for example, lengths, widths, possibly heights, geographic (spatial) locations, etc.

```
type
   LEN, WID, HEI, LOC, ...
value
   attr_LEN: L → LEN
   +: LEN × LEN → LEN
   >: LEN × LEN → Bool
   attr_WID: L → WID
   attr_HEI: L → HEI
   attr_LOC: L → LOC
   ...
```

●

4.2.3 Describing Actions 152

Function Names Actions potentially change states. Actions are here considered deliberate phenomena in that they are caused by willful applications (by agents within the domain being described) of functions having a specific, deliberate purpose, i.e., state change in mind. 153

Example 30 (Transport Net Action Names) Some examples are: *create* an *empty net* (no hubs, no links); *insert* a (new) hub; *insert* a (new) *link* (between a pair of distinct hubs of the net); *delete* (existing) *hub* (having no links into or out from it); and *delete* (existing) *link*.

●

154

Informal Function Descriptions The above examples just listed some actions by their function names. We did not describe these functions. We now do so, for two of these functions.

Example 31 (Informal Transport Net Action Descriptions) We detail two informal function descriptions. The *create empty net* function

47. applies to nothing and yields a net, n, of no hubs and no links.

The *insert link* function

48. applies to a net, n, of at least two distinct hubs, as identified, hi_j, hi_k, by the function application arguments, and a new link ℓ not in n, and yields a net, n′. 155

    a The inserted link l is to be connected to the two distinct hubs identified by hj_i and hk_i, and these designate hubs of the net.

b l is not in the original net.

c The mereology of l designate hj_i and hk_i.

d The state of these identified hubs do not allow any traffic.

e No new hubs are added, hence the set of hub identifiers of the net is unchanged.

f Only one link is added to the net, hence the set of net link identifiers is changed by only the addition of the l link identifier.

g Let the hubs identified by hj_i and hk_i be hj and hk, respectively, before the insertion,

h and by hj' and hk' after the insertion.

i Now the mereology contributions by the two changed hubs reflect only the addition of link l.

156 We leave the informal descriptions of

49. *delete_L*
50. *insert_H*
51. *delete_H*

to the reader.

•

157 **Formal Function Descriptions** We observe actions, but describe the functions which when applied amount to actions. There are two parts to describe a function: (i) the function signature, a:A→B: a distinct function name, say f, and a function type, A→B, that is, type of arguments A and type of results B, and (ii) the function definition, f(a,b)≡$\mathcal{C}$(a): a symbolic function invocation, f(a,b), and a definition body, $\mathcal{C}$(a). $\mathcal{C}$(a) is a clause, i.e.,
158 an expression in FDL, whose evaluation yields the function value.

There are other ways than:

```
value
   f: A → B
   f(a,b) ≡ 𝒞(a)
```

in which to define a function. For example:

```
value
   f: A → B
   f(a) as b
      pre  𝒫(a)
      post 𝒬(a,b)
```

159

Example 32 (Formal Transport Net Action Descriptions) The *create net* function:

```
value
47.  create_N: Unit → N
47.  create_N() as n
47.     post obs_HS(n)={} ∧ obs_LS(n)={}
```

160

The *insert link* function:

```
value
48.  insert_L: (HI × HI) × L → N → N
48.  insert_L((hij,hik),l)(n) as n′
48a.    pre  hji≠hki ∧ {hji,hki} ⊆ xtr_HIs(n) ∧
48b.         ∧ l ∉ obs_LS(n)
48c.         ∧ mereo_L(l) = {hji,hki}
48d.         ∧ attr_HΣ(get_H(hji))(n)={}=attr_HΣ(get_H(hki))(n)
48e.    post xtr_HIs(n) = xtr_HIs(n′)
48f.         ∧ xtr_LIs(n′) = xtr_LIs(n) ∪ {uid_LI(l)}
48g.         ∧ let hj=get_H(hij)(n), hk=get_H(hik)(n),
48h.               hj′=get_H(hij)(n′),vhk′=get_H(hik)(n′) in
48i.         ∧ mereo_H(hj′) = mereo_H(hj) ∪ {uid_LI(l)}
48i.         ∧ mereo_H(hk′) = mereo_H(hk) ∪ {uid_LI(l)} end
```

161

52. From the postulated observer and attribute functions one can define the auxiliary *get* function:

```
value
52.  get_H: HI → N ~→ H
52.  get_H(hi)(n) ≡
52.     let h:H•h ∈ obs_HS(n)∧uid_HI(h)=hi
52.     in h end
52.     pre hi ∈ xtr_HIs(n)
```

162

We do not narrate the informal description of "remaining" net actions (cf. Items 49– 51 on the preceding page), just their function signatures and pre-conditions.

```
49.  delete_L: LI → N ~→ N
49.     pre delete_L(li): li ∈ xtr_LIs(n)
50.  insert_H: H → N ~→ N
50.     pre insert_H(h): h ∉ obs_HS(n)∧mereo_H(h)={}∧mereo_Ω(h)={{}}
51.  delete_H: HI → N ~→ N
51.     pre delete_H(hi): hi ∈ xtr_HIs(n)∧mereo_H(get_H(hi))(n)={}
```

Given appropriate post-conditions the following theorems must be provable:

theorems:

$\forall$ h:H • **pre**−conditions are satisfied $\Rightarrow$ delete_H(uid_HI(h))(insert_H(h)(n))=n
$\forall$ l:L • **pre**−conditions are satisfied $\Rightarrow$ delete_L(uid_LI(l))(insert_L(l)(n))=n

•

163

Agents An *agent* is a behaviour which invokes functions, hence cause actions. So we simply "equate" agents with behaviours.

4.2.4 Describing Events

164

We observe events. But we describe logical properties characterising classes of "the same kind" of events.

Deliberate and Inadvertent (Internal and External) Events Events are like actions: somehow a function was applied either *deliberately* by an agent outside (that is, *external* to) the domain being described, or *inadvertently* by a behaviour of (that is, *internal* to) the domain, but for another purpose than captured by the event. 165

Example 33 (A Deliberate [External] Event) We narrate a simple external cause / "internal" effect example: When one or more bank customers default on their loans and declare themselves unable to honour these loans then the bank may go bankrupt.

•

166

Example 34 (An Inadvertent [Internal] Action Event) We narrate a simple internal cause / "internal" effect example: When a bank customer, the agent, withdraws monies from an account the balance of that account, if the withdrawal is completed, may go negative, or may go below the credit limit. In either case we say that, the withdrawal action, as was intended, succeeded, but that an *"exceeded credit limit"* event occurred.

•

167

Event Predicates Instead of describing events by directly characterising the deliberate external, respectively inadvertent internal actions we suggest to describe these events indirectly, by characterising the logical effects, say, in terms of predicates over *before/after* states. 168

Example 35 (Formalisation of An External Event) The event is that of a *"link segment disappearance"*.

53. Generally we can explain *"link segment disappearances"*, for example, as follows:

54. A l_i-identified link, l, between hubs hf and ht (identified in l) is *removed*.

55. Two hubs, *hf*″ and *ht*″, and two links, *lf* and *lt*, are *inserted* — where
    a hub values *hf* and *ht* (the hubs in the original net) become hub values *hf'* and *ht'* in the resulting net, that is, hub values *hf* and *ht* have same respective hub identifiers as *hf*′ and *ht*′,
    b hubs *hf*″ and *ht*″ are new,
    c links *l*′ and *l*″ are new,
    d link *lf* is *inserted* between *hf*′ and *hf*″, that is, link *lf* identifies hubs *hf*′ and *hf*″, and link *lt* is inserted between *ht*′ and *ht*″, that is, link *lt* identifies hubs *ht*′ and *ht*″, 169
    e hub *hf*′ is, in the resulting net, connected to the links hub *hf* was connected to in the original net "minus" link *l* but "plus" link *lf*,
    f hub *ht*′ is, in the resulting net, connected to the links it was connected to in the original net "minus" link *l* but "plus" link *lt*,
    g hub *hf*″ is connected only to link *lf*
    and hub *ht*″ is connected only to link *lt*,
    h the state space of *hf*′ suitably includes all the possibilities of entering link *lf*[23],
    i the state space of *ht*′ suitably includes all the possibilities of entering link *lt*[24],
    j the state spaces of *hf*″ and *ht*″ both contains just the empty set,
    k the states of *ht*″ and *ht*″ are both the empty set: "dead ends !",
    l the sum of the lengths of links *lf* and *lt* is less than the length of link *l*, and
    m all other non-mereology attributes of *lf* and *lt* are the same as those of link *l*.

56. All other links and hubs are unchanged.

170

value
```
53.  link_segment_disappearance: N × N → Bool
53.  link_segment_disappearance(n,n′) ≡
53.     ∃ l:L, hf″,ht″:H, lf,lt:L •
54.        {l} = obs_LS(n) \ \obs_LS(n′)
55a.         ∧ let hfi=uid_HI(hf), hti=uid_HI(ht) in
55a.            let hf′=get_H(hfi)(n′), ht′=get_H(hti)(n′) in
55b.            {hf″,ht″}∩ obs_HS(n)={} ∧ {hf″,ht″}⊆obs_HS(n′)
55c.         ∧ {lf,lt}∩ obs_LS(n)={} ∧ {lf,lt}⊆obs_LS(n′)
55d.         ∧ mereo_L(l′)={hfi,uid_HI(hf″)} ∧ mereo_L(l″)={hti,uid_HI(ht″)}
55e.         ∧ mereo_H(hf)=mereo_H(hf′)\{uid_LI(l)}∪{uid_LI(lf)}
55f.         ∧ mereo_H(ht)=mereo_H(ht′)\{uid_LI(l)}∪{uid_LI(lt)}
```

[23] A substitution function replaces all link *l* identifiers with link *lf* identifiers.
[24] A substitution function replaces all link *l* identifiers with link *lt* identifiers.

55g. $\wedge$ mereo_H(hf″)={uid_LI(lf)} $\wedge$ mereo_H(ht″)={uid_LI(lt)}
55h. $\wedge$ attr_HΩ(hf′)=subst(uid_LI(l),uid_LI(lf),attr_HΩ(hf))
55i. $\wedge$ attr_HΩ(ht′)=subst(uid_LI(l),uid_LI(lt),attr_HΩ(ht))
55j. $\wedge$ attr_HΩ(hf″)={{}} $\wedge$ attr_HΩ(ht″)={{}}
55k. $\wedge$ attr_HΣ(hf″)={} $\wedge$ attr_HΣ(ht″)={}
55l. $\wedge$ attr_LEN(lf)+attr_LEN(t)<attr_LEN(l)
55m. $\wedge$ $\forall$ X:non_mereo_attributes(l)•attr_X(lf)=attr_X(lt)=attr_X(l)[25]
56. $\wedge$ obs_HS(n′)\{hf′,hf″,ht′,ht″} = obs_HS(n)\{hf,ht}
56. $\wedge$ obs_LS(n′)\{lf,lt} = obs_LS(n)\{l} **end end**

171 We can express a theorem relating the above to the *rem*ove and *ins*ert functions.

theorem: link_segment_disappearance(n,n′) ⇒
    let l:L, hf″,ht″:H, lf,lt:L • [Lines 54–55, 55a–55m, 56] **in**
    n′ = ins_L(lt)(ins_L(lf)(ins_H(ht″)(ins_H(hf″)(rem_L(uid_LI(l))(n)))))
    end

•

4.2.5 Describing Behaviours 172

Behaviour Description Languages As for the description of parts, actions and events[26] there exists formal ways of describing behaviours as of sequences of actions, events and behaviours: some are "textual"[27]: `CSP` [26], some are "graphical", for example: `MSC` [Message Sequence Charts] [27], `Petri Nets` [40] and `State Charts` [25].

173 **Simple Sequential Behaviours:** A simple sequential behaviour is a sequence of actions and events.

— ***Snapshot Description of a Simple Sequential Behaviour:*** Snapshot of a behaviour, as it unfolds, could be described:

let σ' = action_1(arg_1)(σ) $\sqcap$ event_1(σ)(σ') **in**
let σ'' = action_1(arg_1)(σ') $\sqcap$ event_1(σ')(σ'') **in**
...
let $\sigma''...'$ = action_1(arg_1)($\sigma'...'$) $\sqcap$ event_1(`sigm′...′a)($\sigma''...'$) **in**
$\sigma''...'$ **end** ... **end end**

[25]The predicate in Line 55m on the preceding page is to be explained.
[26] Part, action and event description languages were first mentioned in the 'Abstract' footnotes 1– 2 on page 5: `Alloy` [28], `CafeOBJ` [19], `Casl` [13] `Event B` [1], `Maude` [32, 12] `RAISE/RSL` [23, 22], `VDM` [7, 8, 17] and `Z` [42].
[27]The languages mentioned in Footnote 26 are textual.

where the internal non-deterministic operator, $\sqcap$, expresses that either its left side or right operand is chosen. The seemingly recursive equation:

$$\textbf{let } \sigma' = \text{act}(\text{arg})(\sigma) \sqcap \text{event}(\text{arg})(\sigma)(\sigma')$$

expresses a further non-determinism: any σ' satisfying the equation is a valid next state. We can name such simple sequential behaviours, for example: P.

Simple Concurrent Behaviours: A simple concurrent behaviour is a set of two or more simple sequential behaviours. 174

We describe a simple concurrent behaviour by a list of named behaviour descriptions separated by the parallel behaviour composition operator $\|$, for example, P$\|$Q$\|$...$\|$R. A variant form is $\|\{\text{P(i)}|\text{i:Index}\bullet\text{predicate(i)}\}$ which expresses 'distribution' of the behaviour composition operator ($\|$) over 'expanded' terms, that is, $\mathsf{P}(\mathsf{i}_j)\|\mathsf{P}(\mathsf{i}_k)\|\ldots\|\mathsf{P}(\mathsf{i}_\ell)$.

Communicating Behaviours: A communicating behaviour is a behaviour which expresses willingness to engage in a (synchronisation and) communication with another communicating behaviour or with the environment. 175

In order to express 'communication' (between behaviours) a notion of an output/input *channel* is introduced with behaviours allowed 'access' to channel (which are therefore shared). In CSP channels are typed with the type of the values that can be output on a channel "between" behaviours. In CSP output of a value (say of expression e) onto channel 176
ch is expressed by the statement ch!e whereas input of a value from channel ch is expressed by the expression ch?.

```
type
    M
channel
    ch:M
value
    S: Unit → Unit,              S() = P() ∥ Q()
    P: Unit → out ch  Unit,      P() ≡ ... ch!e ...
    Q: Unit → in ch Unit,        Q() ≡ ... ch? ...
```

We thus describe a communicating behaviour by allowing one or more clauses: statements of the kind ch!e and expressions of the kind ch?.

External Non-deterministic Behaviours: A behaviour, $\mathcal{P}$, composed from behaviours 177
$\mathcal{P}_i, \mathcal{P}_j, \ldots, \mathcal{P}_k$ is said to exhibit internal non-determinism if the behaviour is either as is behaviour $\mathcal{P}_i$, or as is behaviour $\mathcal{P}_j$, ..., or as is behaviour $\mathcal{P}_k$, and is influenced in being so by the environment of behaviour $\mathcal{P}$.

We describe such behaviours as follows: P: P_i $\sqcap$ P_j $\sqcap$ ... $\sqcap$ P_k where P_i (etcetera) describes behaviour $\mathcal{P}_i$ (etcetera). A variant description of internal non-determinism is
$\sqcap\{\text{P(i)}|\text{i:Index}\bullet\text{predicate(i)}\}$. 178

External influence is, for example, expressed if behaviour descriptions P_i (etcetera) contain either an output (ch!e) or an input (ch?) clause and the environment offers to accept input, respectively offers output "along" the name channel.

179 **Internal Non-deterministic Behaviours:** A behaviour, $\mathcal{P}$, composed (somehow) from behaviours $\mathcal{P}_i, \mathcal{P}_j, \ldots, \mathcal{P}_k$ is said to exhibit internal non-determinism if the behaviour is either as is behaviour $\mathcal{P}_i$, or as is behaviour $\mathcal{P}_j$, ..., or as is behaviour $\mathcal{P}_k$, and is not influenced in being so by the environment of behaviour $\mathcal{P}$.

We describe such behaviours as follows: P: P_i ⌈⌉ P_j ⌈⌉ ... ⌈⌉ P_k where P_i (etcetera) describes behaviour $\mathcal{P}_i$ (etcetera). A variant description of internal non-determinism is
180 ⌈⌉{P(i)|i:Index•predicate(i)}.

For internal non-determinism to work for expressions like the above we must assume that they do not contain such output (ch!e) or an input (ch?) clauses for which the environment may accept input, respectively offer output.

181 **General Communicating Behaviours:** A general communicating behaviour is a set of sequences of actions, events and (simple sequential, simple concurrent, communicating and non-deterministic) behaviours such that at least two separately identifiable behaviours
182 of a set share at least one channel and contain respective ch!e and ch? clauses.

Example 36 (A Road Pricing (Transport) System Behaviour) This example is quite extensive.

57. A road pricing (transport) system, Δ_{RPS} contains
    a a net n — as outlined in earlier examples — of hubs and links,
    b a fleet f of vehicles and
    c a central road pricing monitor m.
58. From Δ_{RPS} we can observe the
    a a net, n:N,
    b a fleet of vehicles, f:F, and
    c a road pricing monitor, m:M.
59. From the net, n:N, we observe
    a a set of hubs and
    b a set of links
60. From the fleet, f:F, we observe
    a a set of vehicles.

183

type
57. Δ_RPS, N, F, M
value
57a. obs_N: Δ_RPS → N
57b. obs_F: Δ_RPS → F
57c. obs_M: Δ_RPS → M
59a. obs_Hs: N → H-**set**
59b. obs_Ls: N → L-**set**
60a. obs_Vs: F → V-**set**

We need "prepare" some part names:

type
57a. n:N
57b. f:F
57c. m:M
59a. hs:Hs=obs_Hs(n)
59b. ls:Ls=obs_Ls(n)
60a. vs:Vs=obs_VSs(f)

184

61. With the road pricing behaviour we associate separate behaviours,
    a one for the net which is seen as the parallel composition of
        i. a set of hub behaviours
        ii. a set of link behaviours;
    b one for the fleet of vehicles which is seen as the parallel composition of
        i. a set of vehicle behaviours;
    c and a central road pricing monitor behaviour.

185

61. road_pricing_system: **Unit** → **Unit**
61. road_pricing_system() ≡ net()‖fleet()‖monitor(...)

61a. net() ≡
61(a)i. ‖ {hub(uid_HI(h))(h)(vis)|h:H•h ∈ hs} ‖
61(a)ii. ‖ {link(uid_LI(l))(l)(vis)|l:L•l ∈ ls}

61b. fleet() ≡
61(b)i. ‖ {vehicle(obs_VI(v))(v)(vp)|v:V•v ∈ vs}

61c. monitor(...) ≡ ...

186 The *vis* arguments of the *hub* and *link* behaviours "carry" the identifiers of current *v*ehicles currently at the hub or on the link. The *vp* argument of the *vehicle* behaviour "carries" the current *v*ehicle *p*osition. The (...) argument of the *monitor* behaviour records the history status of all vehicles on the net. We omit details of how these arguments are initialised. 187

62. We associate channels as follows:
    a one for each pair of vehicles and hubs,
    b one for each pair of vehicles and links and
    c one for monitor (connected to vehicles).

62. **channel**
62a. $\{$vh_ch[uid_VI(v),uid_HI(h)]$|$v:V,h:H•v $\in$ vd$\wedge$h $\in$ hs$\}$:VH_Msg
62b. $\{$vl_ch[uid_VI(v),uid_LI(h)]$|$v:V,l:L•v $\in$ vs$\wedge$h $\in$ ls$\}$:VL_Msg
62c. mon:VM_Msg

188 We omit detailing the channel message types.

63. Vehicles are positioned
    a either on a link, in direction from one hub to a next, some fraction down that link,
    b or at a hub, in direction from one link to a next where
    c the fraction is a real between 0 and 1.

type
63. VP = onL | atH
63a. onL == mk_onL(li:LI,fhi:HI,f:FRA,thi:HI)
63b. atH == mk_atH(hi:HI,fli:LI,tli:LI)
63c. FRA = **Real axiom** $\forall$ fra:FRA•0$\leq$fra$\leq$1

189

64. The *vehicle* behaviour is modelled as a `CSP` process which communicates with hubs, links and the monitor.

65. The *vehicle* behaviour is a relation over its position.
    If on a link, at some position,
    a then the vehicle may "remain" at that position,
    b chosen so internally non-deterministically,
    c or, if the vehicle position is not "infinitesimally" close to the "next" hub,
    d then the vehicle will move further on along the link,
    e some small fraction δ,
    f else the vehicle moves into the next hub in direction of the link named *li′*
    g where *li′* is in the set of links connected to that hub —
    h while notifying the link, the hub and the monitor of its entering the link and entering the hub.

190

```
value
65e.  $\delta$:Real axiom $0<\delta \ll 1$vehc6
64.   vehicle: VI $\rightarrow$ V $\rightarrow$ VP $\rightarrow$
64.         out,in {vl_ch[ vi,li ]|li:LI•li $\in$ xtr_LIs(ls)} out m_ch Unit
65.   vehicle(vi)(v)(vp:mk_onL(li,fhi,f,thi)) $\equiv$
65a.      vehicle(vi)(v)(vp)
65b.      $\sqcap$
65c.      if f + $\delta$<1
65d.         then vehicle(vi)(v)(mk_onL(li,fhi,f+$\delta$,thi))
65e.         else let li′:LI•li′ $\in$ mereo_H(get_H(thi)(n)) in
65g.                vh_ch[ vi,thi ]!enterH ‖ vl_ch[ vi,li ]!leaveL ‖ m_ch!leaveL_enterH(vi,li,thi);
65h.                vehicle(vi)(v)(mk_atH(thi,li,li′)) end end
```

191

66. If the vehicle is at a hub,
    a then the vehicle may “remain” at that same position,
    b chosen so internally non-deterministically,
    c or move on to the next link,
    d in direction of a next hub,
    e while notifying the hub and monitor of leaving the hub and the link and the monitor of entering the link.

```
66.  vehicle(vi)(v)(vp:mk_atH(hi,fli,tli)) $\equiv$
66a.     vehicle(vi)(v)(vp)
66b.     $\sqcap$
66d.     let {hi′,thi}=mereo_L(getL(tli)(n)) in assert: hi′=hi
66e.     vh_ch[ vi,hi ]!leaveH ‖ vl_ch[ vi,tli ]!enterL ‖ m_ch!leaveH_enterL(vi,hi,tli);
66c.     vehicle(vi)(v)(ml_onL(tli,hi,0,thi)) end
```

192

67. The monitor behaviour records the (dynamic) history of all vehicles on the net: alternating sequences of hub and link identifiers.
68. The monitor contains a price table which to every link and hub records the fee for moving along that link or hub.

```
type
67.  VW′ = VI $\overrightarrow{m}$ (HI|LI)$^*$
67.  VW = {|vw:VW′•wf_VW(vh)|}
68.  Fee, PT = (LI|HI) $\overrightarrow{m}$ Fee
```

```
value
67.  wf_VW(vh) ≡
67.    ∀ hll:(HI|LI)* • hll ∈ rng vh
67.      ∀ i:Nat•{i,i+1}⊆inds hll ⇒
67.        is_HI(hll(i))∧is_LI(hll(i+1))∨is_LI(hll(i))∧is_HI(hll(i+1))
```

193

69. The monitor behaviour non-deterministically externally alternates between
    a input of messages from vehicles
       i. either when entering a link in which case the vehicle history is updated with that link's identifier (for that vehicle),
       ii. or when entering a hub in which case the vehicle history is updated with that hub's identifier (for that vehicle).
    b and accepting inquiries and requests relating vehicle histories and fees (designated by (. . .) below).

194

```
value
69.      monitor: PT → VH → in m_ch  Unit
69.      monitor(pt)(vh) ≡
69a.       (case m_ch? of
69(a)i.      leaveH_enterL(vi,hi,li) → monitor(pt)(vh † [ vi ↦ vh(vi)^⟨li⟩ ])
69(a)ii.     leaveL_enterH(vi,li,hi) → monitor(pt)(vh † [ vi ↦ vh(vi)^⟨hi⟩ ])
69a.        end)
69b.       ⌷ (...)
```

195 We omit description of other monitor actions (Line 69b).

70. Link behaviours maintain a state which records the set of vehicles "currently" on the link.

71. The link behaviour expresses willingness to
    a accept messages from vehicles
    b entering links in which case the *"vehicle vi on link"* state has *vi* added, or
    c leaving links in which case the *"vehicle vi on link"* state has *vi* removed,
    d where these vehicles range over all fleet vehicles.

196

```
type
70.  VIS = VI-set
value
71.  link: li:LI → L → VIS → in cl_Unitlink1
```

```
71.   link(li)(l)(vis) ≡
71a.      ⌷ {let m = cl_vl[vi,li′]? in assert: li′=li
71a.          case m of
71b.            enterL → link(li)(l)(vis ∪ {vi})
71c.            leaveL → link(li)(l)(vis \ {vi})
71d.          end | vi:V•v ∈ xtr_VIs(vs) end}
```

We leave it to the reader to suggest a *hub* behaviour description.

●

4.3 Temporal Issues 197

4.3.1 Three Abstract Time Concepts

We shall briefly examine three aspects of time: *time*, t:T, *absolute time intervals*, ati:(ft:T,tt:T):ATI and *relative* (non-zero) *time intervals*, rti:RTI (where rti is some time interval from any time tf:T to some time tt:T "later").

```
type
    T
    ATI = T × T axiom ∀ (ft,tt):ATI • tt>ft
    TI, RTI
value
    +: T × RTI → T
    −: T × T ~→ RTI pre t−t′: t>t′
    >,=: T × T → Bool
    absolute_to_relative_TI: ATI → RTI
    absolute_to_relative_TI(ta,tb) ≡ tb−ta
```

198 The concept of time has at least two variants: the time process for which time continually increases, and a time descriptor which is a name for time. When we, colloquially say *the time is now* we mean to refer to the process notion; and when we say *the train departs at time* `so-and-so` we mean to refer to the time descriptor notion.

We model the time process notion as a behaviour; and the timenotion descriptor notion as an attribute.

Absolute and relative time intervals are descriptors (i.e., attributes).

4.3.2 Concrete Time Concepts 199

Usually, when speaking of time, we say such things as: *the time is* `12:34` *o'clock* but mean to say the more correct `12:34 pm, May 1, 2012`. An absolute time interval, (t, t'), starts at some time, t, and ends some time, t', thereafter. The time interval $t' - t$ only make sense is $t' > t$. We may include the zero time interval: $t - t$. Several different absolute time intervals may represent the same relative time interval. Two absolute time intervals, (t_a, t_b) and (t_c, t_d), define the same relative time interval iff $t_b - t_a = t_d - t_e$.

4.3.3 Some Interval Relations 200

Two absolute time intervals, (t_a, t_b) and (t_c, t_d), enjoy either of the following relations:

```
value
    =: ATI × ATI → Bool
    (ta,tb) = (tc,td) ≡ ta=tc ∧ tb=td

    prefix: ATI × ATI → Bool
    prefix((ta,tb),(tc,td)) ≡ ta=tc ∧ tb<td

    suffix: ATI × ATI → Bool
    suffix((ta,tb),(tc,td)) ≡ ta>tc ∧ tb=td

    embed: ATI × ATI → Bool
    embed((ta,tb),(tc,td)) ≡ ta<tc ∧ tb>td

    etcetera.
```

4.3.4 Time Phenomena 201

Parts and Time Time descriptors may occur (as components of attributes) in parts.

Example 37 (Train Time Tables) Our example shows a classical use of time descriptors.

72. Trains and stations have names.

73. A train system time table maps train names into train ride descriptions.

74. Train ride descriptions are sequences of two or more station visits

    a such that "later" station visits succeed, in time, those of earlier station visits, and

    b such that station arrival times precede those of same station, same visit departure times.

75. A station visit is a triple: an arrival time, a station name and a departure time.

202

```
type
72. TN, SN
73. TSTT = TN →m TRD
74. TRD′ = SV*,            TRD  = {|trd•wf_TRD(trd)|}
75.  SV′  = T × SN × T,   SV   = {|sv•wf_SV(sv)|}
value
```

```
74a. wf_TRD: TRD′ → Bool
74a. wf_TRD(trd) ≡
74a.     len trd≥2 ∧
74a.     ∀ i:Nat•{i,i+1}⊆inds trd ⇒
74a.         let (_,sn,dt)=trd(i), (at,sn′,_)=trd(i+1) in
74a.         sn≠sn′ ∧ dt<at end
74b. wf_SV: SV′ → Bool
74b. wf_SV(at,_,dt) ≡ at<dt
```

The occurrences of time in the part time table are as part of static attributes of that table. •

203

Actions and Time Actions, "in actual life", takes time. That is, over a(n absolute) time interval. But we abstract from this fact. When such an abstractions becomes unreasonable, that is, when the abstraction distorts a factual representation of the domain, then we shall model the action as a behaviour, typically composed from a sequence of "smaller", i.e., constituent "actions". 204

Still, actions do take place at specific times. So we could decide to represent this temporal aspect of action by respective time stamps. Here we must distinguish between the action itself, i.e., its state change and the description of this action. The former can be represented by a pair of (before,after) states. The latter by a description of the function, the non-state arguments when applied, and the state "in" which it was applied. 205

We decide, for the moment, to not be too specific about this issue. The reason for this is pragmatics: The actions that we do describe (informally and formally) usually represent planned, deliberate phenomena. The fact that they are then "performed" at a certain time is therefore of less importance than is the desired state change. Should the time at which the action is performed influence the result (including state change) then we decide to let that time be an explicit argument of the function invocation that leads to the action. But not all actions are deliberate. Actions that are not planned and deliberate phenomena then represent erroneous actions, that is, actions which the agent has provided with erroneous arguments. For such actions time stamps can be very useful.

Time arguments are then obtained from a separate, you can call it a global, *time behaviour*, one that is always active.

206

Events and Time There are two pragmatic aspects which separate actions from events. Firstly, in contrast to actions, the times at which events occur appear to be important.Event occurrences are basically spontaneous: not planned for. Therefore it may be important to record the time of their occurrence. Secondly, we take the view that actions are primarily characterised by their effect on the state, that is, we emphasise an explicit description of the state change, whereas events appears to be primarily characterisable in terms of properties of the event, that is, we emphasise a predicate description of the state change.

Behaviours and Time It can be relevant to record time in connection with behaviours. 207

Example 38 (Timed Monitor Behaviour) We modify our monitor behaviour.

76. Vehicle histories now record the time at which vehicles enter and leave hubs and links.

```
type
76.  Time
76.  $VH' = VI \underset{m}{\rightarrow} (Time \times (HI|LI))^*$
76.  $VH = \{|vh:VH' \bullet wf\_WH(wh)|\}$
value
76.  $wf\_VH: VH' \rightarrow$ **Bool**
76.     $\forall hll:(T \times (HI|LI))^* \bullet hll \in$ **rng** vh
76.        $\forall i:$**Nat**$\bullet\{i,i+1,i+2\} \subseteq$ **inds** hll $\Rightarrow$
76.           **let** $(t,hl)=hll(i), (t',hl')=hll(i+1), (t'',hl'')=hll(i+2)$ **in**
76.           $t<t'<t''$
76.         $\wedge$ $(is\_HI(hll(i)) \wedge is\_HI(hll(i+1)) \wedge is\_LI(hll(i+2))$
76.           $\vee$ $is\_LI(hll(i)) \wedge is\_HI(hll(i+1)) \wedge is\_HI(hll(i+2))$
76.           $\vee$ $is\_LI(hll(i)) \wedge is\_LI(hll(i+1)) \wedge is\_HI(hll(i+2)))$
76.           **end**
```

208

77. The *monitor* behaviour now, additionally,
    a interacts with a (global) *clock* behaviour
    b over a channel *clock_ch*.
    c The clock internally non-deterministically issues the current time or skips that point while similarly internally non-deterministically either not stepping the clock, or stepping it
    d with an "infinitisimal" time interval.

78. With the introduction of a clock we need redefine the system context.

209

```
channel
77b.  **channel** clock_ch: T
value
77.   monitor: PT → VH → **in** m_ch; **in**,**out** clock_ch **Unit**

77a.  clock: **Unit** → **out** clock_ch **Unit**
77c.  clock(t) ≡ (**skip** ⌈⌉ clock_ch!t) ; (clock(t ⌈⌉ (t+ti)))

77d.  ti:TI **axiom:** ti is a tiny time interval

78.   timed_system() = road_pricing_system() ∥ clock(t)
```

210

79. When accepting messages from a link or a hub the *monitor* inquires with the *clock* as to *“what time is it?”* by expressing willingness to input that time.

80. That time is recorded both as the time the vehicle leaves the hub and as the time it enters the link.

81. Similar for leaving links and entering hubs.

```
value
77.   monitor(pt)(vh) ≡
69a.    (case m_ch? of
79.        let t = clock_ch? in
80.        leaveH_enterL(vi,hi,li) → monitor(pt)(vh † [ vi ↦ vh(vi)⌢⟨(t,hi),(t,li)⟩ ])
81.        leaveL_enterH(vi,li,hi) → monitor(pt)(vh † [ vi ↦ vh(vi)⌢⟨(t,li),(t,hi)⟩ ])
69a.     end end)
69b.    ⌷ (...)
```

The timed behaviour implied in the Road Pricing Transport System Behaviour with the Timed Monitor Behaviour can be made more explicit. •

211

Example 39 (Traffic Behaviours) We abstract further.

82. Traffic can be abstracted as a
   a continuous or
   b discrete)

   function from time to pairs of nets[28] and vehicle positions.

   c Vehicles are positioned either at hubs or on links.
   d Vehicle hub positions, in principle, need only be represented by the hub identifier.
   e Vehicle link positions can be represented by the link identifier and a pair of the fraction “down” the link from a hub, here identified by that hub's hub identifier.

212

```
type
82a.  cTF = T → (N × (V →ₘ VP))
82b.  dTF = T →ₘ (N × (V →ₘ VP))
82c.  VP  = atH | onL
82d.  atH :: HI
82e.  onL :: LI × (FRA × HI)
```

[28]As nets may change due, for example, to insertion and removal of hubs and links.

Instead of modelling vehicle positions in terms of actual vehicles, v:V, we can model vehicle positions in terms of vehicle identifiers, vi:VI. (It all depends on the use of the traffic abstracts, cTF and dTF.) Similarly for choosing continuous or discrete time models. ●

213

Example 40 (Traffic Abstraction) We can introduce a behaviour, *traffic*, which, based on communications with a modified *vehicle* behaviour, "builds" a discrete traffic. We start by redefining the *vehicle* behaviour shown on Page 59.

83. The *traffic* behaviour accepts inputs from the *vehicle* behaviour over a shared channel *trf_ch*.
84. Whenever the *vehicle* is invoked a message is communicated to the *traffic* behaviour with the current vehicle position.

214

```
channel
83.  trf_ch:(VI×VP)
value
64.   vehicle: VI → V → VP →
83.     out,in {vl_ch[vi,li]|li:LI•li ∈ xtr_LIs(ls)} out m_ch,trf_ch Unit
65.   vehicle(vi)(v)(vp:mk_onL(li,fhi,f,thi)) ≡
84.     (trf_ch!(vi,vp);
65a.     vehicle(vi)(v)(vp))
65b.     ⌈⌉
65c.     if f + δ<1
84.        then (trf_ch!(vi,mk_onL(li,fhi,f+δ,thi)) ;
65d.           vehicle(vi)(v)(mk_onL(li,fhi,f+δ,thi)))
65e.        else let li′:LI•li′ ∈ mereo_LIs(get_H(thi)(n)) in
65g.              vh_ch[vi,thi]!enterH ∥ vl_ch[vi,li]!leaveL ∥ m_ch!leaveL_enterH(vi,li,thi)
84.             ∥ trf_ch!(vi,mk_atH(thi,li,li′));
65h.              vehicle(vi)(v)(mk_atH(thi,li,li′)) end end
```

215

```
value
64.   vehicle: VI → V → VP →
83.     out,in {vh_ch[vi,hi]|hi:HI•hi ∈ xtr_HIs(hs)} out m_ch,trf_ch Unit
66.  vehicle(vi)(v)(vp:mk_atH(hi,fli,tli)) ≡
84.     (trf_ch!(vi,vp);
66a.     vehicle(vi)(v)(vp))
66b.     ⌈⌉
66d.     let {hi′,thi}=mereo_HIs(getL(tli)(n)) in assert: hi′=hi
66e.     vh_ch[vi,hi]!leaveH ∥ vl_ch[vi,tli]!enterL ∥ m_ch!leaveH_enterL(vi,hi,tli)
84.     ∥ trf_ch!(vi,ml_onL(tli,hi,0,thi));
66c.     vehicle(vi)(v)(ml_onL(tli,hi,0,thi)) end
```

216

85. Traffic is simplied in not considering the otherwise possibly dynamically changing net.
86. The *traffic* behaviour, when accepting input from some vehcile, *vi*, as to its position, *vp*, obtains, the current time, *t*, from the *clock* behaviour.
87. A technicality: if there are recordings for some vehicles in the traffic, *trf*, at that tine,
88. then *trf* is modified in one way,
89. otherwise it is modified in another way.

217

```
type
85. TRF = T $\overrightarrow{m}$ (VI $\overrightarrow{m}$ VP)
value
83. traffic: TRF → in trf_ch, clock_ch  Unit
83. traffic(trf) ≡
86.    let (vi,vp) = trf_ch?, t = clock_ch? in
87.    if t ∈ dom trf
88.       then trf † [t ↦ trf(t) † [vi ↦ vp]]
89.       else trf ∪ [t ↦ [vi ↦ vp]] end end
```

4.3.5 Temporal Descriptions

218

MORE TO COME

Examples of temporal description languages are: the `Interval Temporal Logic (ITL)` [36, 16, 37], the `Duration Calculus (DC)` [43], and the `Temporal Logic of Actions`$^+$ `(TLA`$^+$`)` [29].

MORE TO COME

5 Discovering Domain Entities 219

Section 2 briefly characterised, informally and also a bit more formally, what we mean by a domain. Section 3 informally and systematically characterised the four categories of entities: parts, actions, events and behaviours. Section 4 more-or-less "repeated" Sect. 3's material but by now giving more terse narratives (that is, informal descriptions) and, for the fist time, also formalisations. Section 4 did not hint at how one discovers domain parts (i.e., their types), actions, events and behaviours. In this section we try unravel a set of techniques and tools — so-called 'discoverers' and 'analysers' — using which the domain describer (scientist and/or engineer) can more-or-less systematically discover, analyse and describe a domain, informally and formally.

5.1 Preliminaries 220

Before we present the discoverers and analysers we need establish some concepts.

5.1.1 Part Signatures 221

Let us consider a part $p : P$. Let $p : P$, by definition, be the *principal part* of a domain. Now we need to identify (i) the type, P, of that part; (ii) the types, $\mathsf{S}_1, \ldots, \mathsf{S}_m$, of its proper sub-parts (if p is composite); (iii) the type, PI, of its unique identifier; (iv) the possible types, $\mathsf{MI}_1, \ldots, \mathsf{MI}_n$, of its mereology; and (v) the types, $\mathsf{A}_1, \ldots, \mathsf{A}_o$, of its attributes. We shall name that cluster of type identifications the *part signature*. We refer to P as identifying the part signature. Each of the S_i (for $i : \{1..m\}$) identifies sub-parts and hence sub-part, i.e., part signatures. 222

Example 41 (Net Domain and Sub-domain Part Signatures) The part signature of the hubs and the links are here chosen to be those of(i) the (root) net type, N, (ii) the (sub-domain) set of hubs type Hs, (ii) the (sub-domain) set of links type Ls and (v) the type of net attributes Net_name, Net_owner, etc. The part signatures Hs and Ls are (ii) Hs = H-**set**, H (iii,iv) HI, LI-**set** (v) Hub_Nm, Location, HΣ, HΩ, ... (ii) Ls = L-**set**, L (iii,iv) LI, HI-**set** (v) Link_Nm, LΣ, LΩ, LEN, etc. •

5.1.2 Domain Indices 223

By a domain index we mean a list of part type names that identify a sequence of part signatures. More specifically The domain Δ has index $\langle\Delta\rangle$. The sub-domains of Δ, with part types A, B, ..., C, has indices $\langle\Delta,A\rangle$, $\langle\Delta,B\rangle$, $\ldots$, $\langle\Delta,C\rangle$. The sub-domains of sub-domain with index ℓ and with part types A, B, ..., C has indices $\ell\widehat{\ }\langle A\rangle$, $\ell\widehat{\ }\langle B\rangle$, $\ldots$, $\ell\widehat{\ }\langle C\rangle$.

224

Example 42 (Indices of a Road Pricing Domain) We refer to the the *Road-pricing Transport Domain*, cf. Example 36 on page 56.

The sub-domain indices of the road-pricing transport domain, Δ, are: $\langle\Delta\rangle$, $\langle\Delta$,N$\rangle$, $\langle\Delta$,F$\rangle$, $\langle\Delta$,M$\rangle$, $\langle\Delta$,N,Vs$\rangle$, $\langle\Delta$,N,Ls$\rangle$, $\langle\Delta$,N,H$\rangle$, $\langle\Delta$,N,L$\rangle$ and $\langle\Delta$,F,V$\rangle$. •

5.1.3 Inherited Domain Signatures 225

Let $\langle\Delta$,A,B,C,D$\rangle$ be some domain index. Then $\langle\Delta$,A,B,C$\rangle$ $\langle\Delta$,A,B$\rangle$ $\langle\Delta$,A$\rangle$ $\langle\Delta\rangle$ are the inherited domain indices of $\langle\Delta$,A,B,C,D$\rangle$.

5.1.4 Domain and Sub-domain Categories 226

By the domain category of the domain indexed by ℓ⌢⟨D⟩ we shall mean the domain signature of D, and the action, event and behaviour definitions whose signatures involves just the types given in the domain signature of D or in inherited domain signatures. 227

Example 43 (**The Road-pricing Domain Category**) The road-pricing domain category consist of the types N, F and M, the create_Net create_Fleet and create_M actions, and corresponding Net, Fleet and M behaviours •

228

By a sub-domain category, of index ℓ, we shall mean the sub-domain types of the sub-domain designated by index ℓ, and the actions, events and behaviours whose signatures involves just the types of the ℓ indexed sub-domain or of any prefix of ℓ indexed sub-domain or of the root domain. 229

Example 44 (**A Hub Category of a Road-pricing Transport Domain**) The ancestor sub-domain types of the hub sub-domain are: *HS*, *N* and Δ. The hub category thus includes the part (etc.) types H, HI, ..., the insert_Hub and the delete_Hub actions, perhaps some saturated_hub (and/or other) event(s), but probably no hub behaviour as it would involve at least the type LI which is not in an ancestor sub-domain of the Hub sub-domain. •

5.1.5 Simple and Compound Indexes 230

By a simple index we mean a domain or a sub-domain index. By a compound index we mean a set of two or more distinct indices of a domain Δ. Compound indices, $c_{idx} : \{\ell_i, \ell_j, \ldots, \ell_k\}$, designate parts, actions, events and behaviours each of whose types and signatures involve types defined by all of the simple indexes of c_{idx}.

Example 45 (**Compound Indices of the Road-pricing System**) We show just one compound index: $\{\langle\Delta$,N,HS,H$\rangle$,$\langle\Delta$,N,LS,L$\rangle\}$. •

5.1.6 Simple and Compound Domain Categories 231

By a simple domain category we shall mean any ℓ-indexed [sub-]domain category. By the compound domain category of compound index $c_{idx} : \{\ell_i, \ell_j, \ldots, \ell_k\}$, we shall mean the set of types, actions, events and behaviours as induced by compound index c_{idx}, that is, parts, actions, events and behaviours each of whose types and signatures involve types defined by all of the simple indexes of c_{idx}. 232

Example 46 (The Compound Domain Category of Hubs and Links) The compound domain category designated by $\{\langle\Delta,N,HS,H\rangle,\langle\Delta,N,LS,L\rangle\}$ includes:

```
type
    HIs = HI-set
        axiom ∀ his:HIs•card his=2
    LIs = LI-set
    HΣ = (LI×LI)-set
    LΣ = (HI×HI)-set
    HΩ = HΣ-set
    LΩ = LΣ-set
value
    mereo_L: L → HIs,
    mereo_H: H → LIs
    attr_HΣ: H → HΣ
    attr_LΣ: L → LΣ
    attr_HΩ: H → HΩ
    attr_LΩ: L → LΩ
axiom
    ∀ h:H•attr_HΣ(h)⊆attr_HΩ(h)
    ∀ l:L•attr_LΣ(l)⊆attr_LΩ(l)
```

•

5.1.7 Examples 233

We repeat some examples, but now "formalised".

Example 47 (The Root Domain Category) We start at the root, Δ, of the Road Pricing Domain. See Fig. 13.

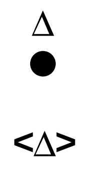

Figure 13: The $\langle\Delta\rangle$ Root

At the root we 'discover' the net, fleet and road pricing monitor. See Fig. 14 on the facing page. 234

235 When observing the very essence of the road pricing domain "at the $\langle\Delta\rangle$ level" one observes:

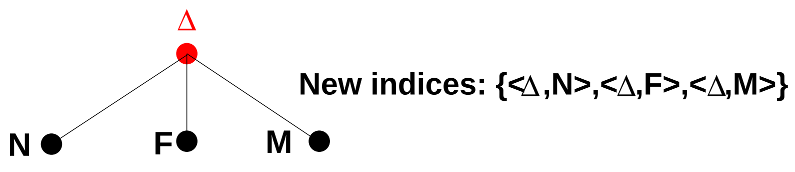

Figure 14: Exploring the root index ⟨Δ⟩ Index

```
type
    N, F, M
value
    obs_N: Δ → N
    obs_F: Δ → F
    obs_M: Δ → M
    attr_...: Δ → ...
```

where ... stands for types of road pricing domain attributes.

● 236

Example 48 (The Net Domain Category) We then proceed to explore the domain at index ⟨Δ,N⟩. See Fig. 15.

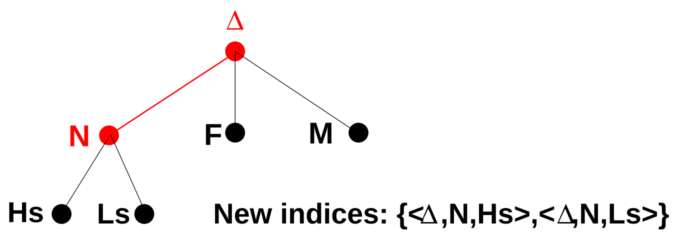

Figure 15: Exploring the ⟨Δ,N⟩ Index

237 When observing the very essence of the Net domain, "at the ⟨Δ, N⟩ level" one observes:

```
type
    Hs = H-set
    Ls = L-set
    H
    L
    ...
value
    obs_Hs: N → Hs
    obs_Ls: N → Ls
    attr_Hs: Hs → ...
    attr_Ls: Ls → ...
```

where ... stand for attributes of the Hs and the Ls parts of N.

•

238

Example 49 (The Fleet Domain Category) We then proceed to explore the domain at index ⟨Δ,F⟩. See Fig. 16.

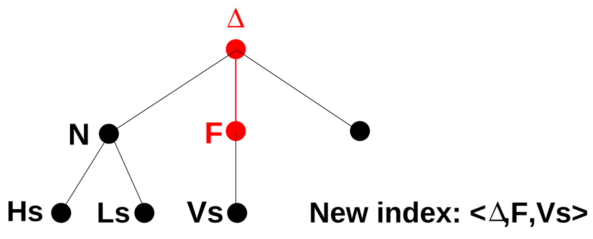

Figure 16: Exploring the ⟨Δ,F⟩ Index

239

When observing the very essence of the Fleet domain, "at the ⟨Δ, F⟩ level" one observes:

```
type
    Vs = V-set
    V
    ...
value
    obs_Vs: F → Vs
    attr_... Vs → ...
```

where ... stand for attributes that we may wish to associate with Fleets of vehicles.

•

240

Example 50 (The Hub Domain Category) We now switch "back" to explore the domain at index ⟨Δ,N,Hs⟩. See Fig. 17 on the next page. 241

When observing the very essence of the Fleet domain, "at the ⟨Δ, N,Hs,H⟩ level" one observes:

```
type
    HI
    ...
value
    uid_HI: H → HI
    attr_...: H → ...
```

where ... stand for LOCation, etc.

•

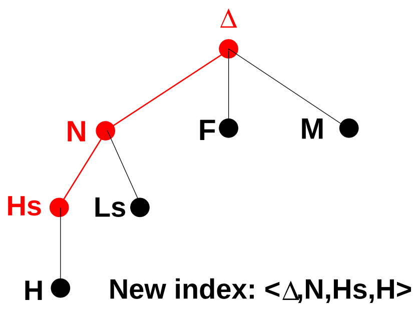

Figure 17: Exploring the ⟨Δ, N,Hs,H⟩ Index

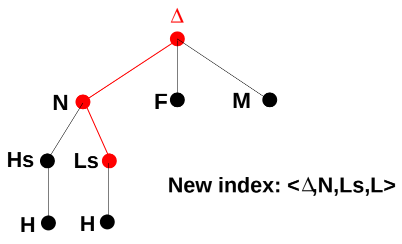

Figure 18: Exloring the ⟨Δ, N,Ls,L⟩ Index

242

243 **Example 51 (The Link Domain Category)** Next we explore the link domain. See Fig. 18.

When observing the very essence of the Fleet domain, "at the ⟨Δ, N,Ls,L⟩ level" one observes:

```
type
    LI
    ...
value
    uid_LI: L → LI
    attr_...: L → ...
```

where ... stand for LOCation, LENgth, etc.

●

244

Example 52 (The Compound Hub and Link Domain Category) We next explore a compound domain. See Fig. 19 on the next page. 245

When observing the very essence of the Fleet domain, at the {⟨Δ, N,Hs,H⟩, ⟨Δ, N,Ls,L⟩} level one observes:

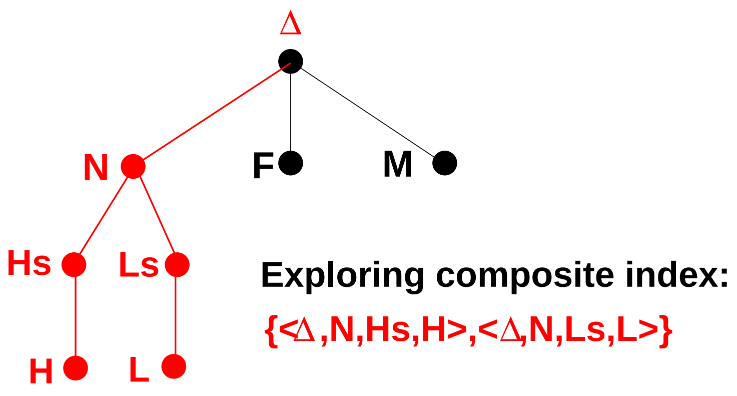

Figure 19: Exploring composite index $\{\langle\Delta, \mathsf{N},\mathsf{Hs},\mathsf{H}\rangle, \langle\mathsf{N},\mathsf{Ls},\mathsf{L}\rangle\}$

type
  HΣ = (LI×LI)-**set**, HΩ = HΣ-**set**,
  LΣ = (HI×HI)-**set**, LΩ = LΣ-**set**
value
  attr_HΣ: H → HΣ, attr_HΩ: H → HΩ
  attr_LΣ: L → LΣ, attr_LΩ: L → LΩ
  mereo_L: L → HI-**set** **axiom** ∀ l:L:**card** mereo_L(l)=2
  mereo_H: H → LI-**set** (= LI-**set**)
  remove_H: HI → N $\xrightarrow{\sim}$ N
  insert_L: L → N $\xrightarrow{\sim}$ N
  remove_L: LI → N $\xrightarrow{\sim}$ N
  ...
axiom
  ∀ hσ:HΣ ..., ∀ hω:HΩ ...
  ∀ lσ:HΣ ..., ∀ lω:HΩ ...

•

5.1.8 Discussion 246

The previous examples (47–52), especially the last one (52), illustrates the complexity of a domain category; from just observing sub-part types and attributes (as in Examples 47–49), Example 52 observations grow to intricate mereologies etcetera. The 'discoverers' that we shall propose aim at structuring the discovery process by focusing, in turn, on part sorts, concrete part types, unique identifier types of parts, part mereology, part attributes, action signatures, event signatures, etc.

5.2 Proposed Type and Signature 'Discoverers' 247

By a 'domain discoverer' we shall understand a tool and a set of principles and techniques for using this tool in the discovery of the entities of a domain.

In this section we shall put forward a set of type and signature discoverers. Each discoverer is indexed by a simple or a compound domain index. And each discoverer is dedicated to some aspect of some entities. Together the proposed discoverers should cover the most salient aspects of domains. Our presentation of type and signature discoverer does not claim to help analyse "all" of a domain. 248

We need formally define what an index is.

type
  Index = Smpl_Idx | Cmpd_Idx
  Smpl_Idx = $\{|\ \langle\Delta\rangle^\frown\mathrm{idx} \mid \mathrm{idx}{:}\mathrm{Type\_Name}^*\ |\}$
  Cmpd_Idx′ = Smpl_Idx-**set**
  Cmpd_Idx = $\{|$ sis:Cmpd_Idx′ • wf_Cmpd_Idx(sis) $|\}$
value
  wf_Cmpd_Idx: Cmpd_Idx′ → **Bool**
  wf_Cmpd_Idx(sis) ≡ ∀ si,si′:Smpl_Idx • {si,si′}⊆sis ∧ si≠si′

  DISCOVERER_KIND: Index → **Text**
  DISCOVER_KIND($\ell^\frown\langle t\rangle$) **as** text
    pre: $\ell^\frown\langle t\rangle$ is a valid index beginning with Δ
    post: text is some, in our case, RSL text

The idea of the $\ell^\frown\langle t\rangle$ index is that it identifies a sub-domain, t, of Δ where *DISCOVERER-_KIND* is any of the several different "kinds" of domain forms: 249

[90 (Page 76)] PART_SORTS,

[91 (Page 77)] HAS_A_CONCRETE_TYPE,

[92 (Page 78)] PART_TYPES,

[93 (Page 79)] UNIQUE_ID,

[96 (Page 79)] MEREOLOGY,

[98 (Page 80)] ATTRIBUTES,

[100 (Page 82)] ACTION_SIGNATURES,

[101 (Page 83)] EVENT_SIGNATURES and

[103 (Page 84)] BEHAVIOUR_SIGNATURES.

In a domain analysis (i.e., discovery) the domain description emerges "bit-by-bit". Initially types are discovered and hence texts which define unique identifier types and functions, mereology types and functions, and attribute types and functions. Then the signatures of actions, events and behaviours.

You may consider these "piece-wise" texts as being "added" to a (hence) growing reservoir of (RSL) texts with this reservoir being continually inspected by the domain analyser.

5.2.1 Analysing Domain Parts 250

The two most important aspects of an algebra are those of its parts and its operations. Rather than identifying, that is, discovering or analysing individual parts we focus on discovering their types — initially by defining these as sorts. And rather than focusing on defining what the operations achieve we concentrate on the signature, i.e., the types of the operations. 251

It (therefore) seems wise to start with the discovery of parts, and hence of their types. Part types are present in the signatures of all actions, events and behaviours. When observing part types we also observe a variety of part type analysers: possible unique identities of parts, the possible mereologies of composite parts, and the types of the attributes of these parts.

252

Domain Part Sorts and Their Observers Initially we "discover" parts — by deciding upon their types, in the form, first of sorts, subsequently and possibly in the form of concrete types.

253 **A Domain Sort Discoverer:**

90. A part type discoverer applies to a simply indexed domain, *index*, and yields
    a a set of type names
    b each paired with a part (sort) observer.

value
90. $\mathbb{PART\_SORTS}$: Index $\overset{\sim}{\rightarrow}$ **Text**
90. $\mathbb{PART\_SORTS}(\ell^\frown\langle T\rangle)$:
90a. tns:$\{T_1,T_2,...,T_m\}$:TN-**set** $\times$
90b. $\{$ obs_T_j: $T \rightarrow T_j \mid T_j$:tns$\}$

254

Example 53 (Some Part Sort Discoveries) We apply a concrete version of the above sort discoverer to the road-pricing transport domain Δ:

$\mathbb{PART\_SORTS}(\langle\Delta\rangle)$:
  type
    N, F, M
  value
    obs_N: $\Delta \rightarrow$ N
    obs_F: $\Delta \rightarrow$ F

```
        obs_M: Δ → M

PART_SORTS(⟨Δ,N⟩):
    type
        Hs, Ls
    value
        obs_Hs: N → Hs
        obs_Ls: N → Ls


PART_SORTS(⟨Δ,F⟩):
    type
        Vs
    value
        obs_Cs: F → Vs
```

●

255

Domain Part Types and Their Observers

Do a Sort Have a Concrete Type ? Sometimes we find it expedient to endow a "discovered" sort with a concrete type expression, that is, "turn" a sort definition into a concrete type definition.

91. Thus we introduce the "discoverer":

```
91  HAS_A_CONCRETE_TYPE: Index → Bool
91  HAS_A_CONCRETE_TYPE(ℓ⌢⟨t⟩):true|false
```

256

Example 54 (**Some Type Definition Discoveries**) We exemplify two true expressions:

```
    HAS_A_CONCRETE_TYPE(⟨Δ,N,Hs⟩)
    HAS_A_CONCRETE_TYPE(⟨Δ,N,Ls⟩)
    ∼ HAS_A_CONCRETE_TYPE(⟨Δ,N,Hs,H⟩)
    ∼ HAS_A_CONCRETE_TYPE(⟨Δ,N,Ls,L⟩)
```

●

A Domain Part Type Observer: The $\mathbb{PART\_TYPES}(\ell\widehat{\ }\langle \mathsf{t}\rangle)$ invocation yields one or more sort definitions of part types together with their observer functions. The domain analyser can decide that some parts can be immediately analysed into concrete types. Thus, together with yielding a type name, the $\mathbb{PART\_TYPES}$ can be expected to yield also a type definition, that is, a type expression (paired with the type name). Not all type expressions make sense. We suggest that only some make sense. 257 258

92. The $\mathbb{PART\_TYPES}$ discoverer applies to a composite type, t, and yields
    a a type definition, $\mathsf{T} = \mathsf{TE}$,
    b together with the sort and/or type definitions of so far undefined type names of TE.
    c The $\mathbb{PART\_TYPES}$ discoverer is not defined if the designated sort is judged to not warrant a concrete type definition.

```
92.   PART_TYPES: Index $\xrightarrow{\sim}$ Text
92.   PART_TYPES($\ell\widehat{\ }\langle$t$\rangle$):
92a.        type t = te,
92b.              $T_1$ or $T_1 = TE_1$
92b.              $T_2$ or $T_2 = TE_2$
92b.              ...
92b.              $T_n$ or $T_n = TE_n$
92c.        pre: HAS_A_CONCRETE_TYPE($\ell\widehat{\ }\langle$t$\rangle$)
```

259

Example 55 (Some Part Type Discoveries) We exemplify two discoveries:

```
PART_TYPES($\langle\Delta$,N,Hs$\rangle$):
      type
            H
            Hs = H-set

PART_TYPES($\langle\Delta$,N,Ls$\rangle$):
      type
            L
            Ls = L-set

PART_TYPES($\langle\Delta$,F$\rangle$):
      type
            V
            Vs = V-set
```

•

260 **Concrete Part Types:** In Example 55 on the facing page we illustrated one kind of concrete part type: sets. Practice shows that sorts often can be analysed into sets. Other analyses of part sorts are Cartesians, list, and simple maps:

```
90b.  te: tn1 × tn2 × ... × tnm
90b.  te: tn*
90b.  te: Token →ₘ tn
```

where tn's are part type – usually sort – names some of which may have already been defined, and where Token is some simple atomic (non-part) type.

Part Type Analysers There are three kinds of analysers: *unique identity* analysers, *mereology* analysers and *general attribute* analysers. and 261

Unique Identity Analysers: We associate with every part type T, a unique identity type TI. 262

93. So, for every part type T we postulate a unique identity analyser function uid_TI.

```
value
93. UNIQUE_ID: Index → Text
93. UNIQUE_ID(ℓ⌢⟨T⟩):
93.     type
93.         TI
93.     value
93.         uid_TI: T → TI
```

Mereology Analysers: We remind the reader of Sects. 3.1.6 on page 32. Given a part, p, of type T, the mereology, MEREOLOGY, of that part is the set of all the unique identifiers of the other parts to which part p is partship-related as "revealed" by the mereo_TI$_i$ functions applied to p. 263

94. Let types T_1, T_2, ..., T_n be the types of all parts of a domain.
95. Let types TI_1, TI_2, ..., TI_n[29], be the types of the unique identifiers of all parts of that domain.
96. The mereology analyser MEREOLOGY is a generic function which applies to an index and yields the set of

    a zero,

[29]We here assume that all parts have unique identifications.

b one or

c more

mereology observers.

264

```
type
94.  T  = T₁ | T₂ | ... | Tₙ
95.  T_idx = TI₁ | TI₂ | ... | TIₙ
96.  𝕄𝔼ℝ𝔼𝕆𝕃𝕆𝔾𝕐: Index → Text
96.  𝕄𝔼ℝ𝔼𝕆𝕃𝕆𝔾𝕐({ℓᵢ⁀⟨Tⱼ⟩,...,ℓₖ⁀⟨Tₗ⟩}):
96a.          either:  {}
96b.          or:         mereo_TIₓ: T → (TIₓ|TIₓ-set)
96c.            or:       { mereo_TIₓ: T → (TIₓ|TIₓ-set),
96c.                          mereo_TI_y: T → (TI_y|TI_y-set),
96c.                          ...,
96c.                          mereo_TI_z: T → (TI_z|TI_z-set) }
```

where none of TI_x, TI_y, . . . , TI_z are equal to TI and each is some T_{idx}.

265 **General Attribute Analysers:** A general attribute analyser analyses parts beyond their unique identities and possible mereologies.

97. Part attributes have names. We consider these names to also abstractly name the corresponding attribute types, that is, the names function both as attribute names and sort names. Finally we allow attributes of two or more otherwise distinct part types to be the same.

98. 𝔸𝕋𝕋ℝ𝕀𝔹𝕌𝕋𝔼𝕊 applies to parts of any part type t and yields

99. the set of attribute observer functions attr_at, one for each attribute sort at of t.

266

```
type
97.  AT = AT₁ | AT₂ | ... | ATₙ
value
98.  𝔸𝕋𝕋ℝ𝕀𝔹𝕌𝕋𝔼𝕊: Index → Text
98.  𝔸𝕋𝕋ℝ𝕀𝔹𝕌𝕋𝔼𝕊(ℓ⁀⟨T⟩):
99.      type
99.          AT₁, AT₂, ..., ATₘ
99.      value
99.          attr_AT₁: T → AT₁
99.          attr_AT₂: T → AT₂
99.          ...,
99.          attr_ATₘ: T → ATₘ, m≤n
```

267

Example 56 (Example Part Attributes) We exemplify attributes of composite and of atomic parts:

```
ATTRIBUTES(⟨Δ⟩):
    type
        Domain_Name, ...
    value
        attr_Name: Δ → Domain_Name
        ...
```

where Domain_Name could include *State Roads* or *Rail Net.* etcetera. 268

```
ATTRIBUTES(⟨Δ,N⟩):
    type
        Sub_Domain_Location, Sub_Domain_Owner, Kms, ...
    value
        attr_Location: N → Sub_Domain_Location
        attr_Owner: N → Sub−Domain_Owner
        attr_Length: N → Kms
        ...
```

where Sub_Domain_Location could include *Denmark*, Sub_Domain_Owner could include *The Danish Road Directorate*[30], respectively *BaneDanmark*[31], etcetera. 269

```
ATTRIBUTES(⟨Δ,N,Hs,L⟩):
    type
        LOC, LEN, ...
    value
        attr_LOC: L → LOC
        attr_LEN: L → LEN
        ...
```

```
ATTRIBUTES({⟨Δ,N,Hs,L⟩.⟨Δ,N,Hs,H⟩}):
    type
        LΣ = HI-set, LΩ − LΣ-set
        HΣ = LI-set, HΩ − HΣ-set
    value
        attr_LΣ: L → LΣ
```

[30] http://www.vejdirektoratet.dk/roaddirectorate.asp?page=dept&objno=1024
[31] http://uk.bane.dk/default_eng.asp?artikelID=931

attr_LΩ: L → LΩ
attr_HΣ: H → HΣ
attr_HΩ: H → HΩ

where LOC might reveal some Bezier curve[32] representation of the possibly curved three dimensional location of the link in question, LEN might designate length in meters, LΣ [HΣ] designates the state of the link [hub], LΩ [HΩ] designates the space of all allowed states of the link [hub], etcetera.

•

270 — ***Attribute Sort Exploration:*** Once the attribute sorts of a part type have been determined there remains to be "discovered" the concrete types of these sorts. We omit treatment of this point in the present version of these these research notes.

5.2.2 Discovering Action Signatures 271

General We really should discover actions, but actually analyse function definitions. And we focus, in these research notes, on just "discovering" the function signatures of these actions. By a function signature, to repeat, we understand a functions name, say fct, and a function type expression (te), say dte$\tilde{\to}$rte where dte defines the type of the function's definition set and rte defines the type of the function's image, or range set.

272

Function Signatures Usually Depend on Compound Domains We use the term 'functions' to cover actions, events and behaviours.

We shall in general find that the signatures of actions, events and behaviours depend on types of more than one domain. Hence the schematic index set $\{\ell_1\widehat{\ }\langle t_1\rangle,\ell_2\widehat{\ }\langle t_2\rangle,...,\ell_n\widehat{\ }\langle t_n\rangle\}$ is used in all actions, events and behaviours discoverers.

273

The ACTION_SIGNATURES Discoverer

100. The ACTION_SIGNATURES meta-function applies to an index set and yields
   a a set of action signatures each consisting of an action name and a pair of definition set and range type expressions where
   b the type names that occur in these type expressions are defined by in the domains indexed by the index set.

100 ACTION_SIGNATURES: Index $\tilde{\to}$ **Text**
100 ACTION_SIGNATURES($\{\ell_1\widehat{\ }\langle T_1\rangle,\ell_2\widehat{\ }\langle T_2\rangle,...,\ell_n\widehat{\ }\langle T_n\rangle\}$):
100a act_fct$_i$: te$_{i_d}$ $\tilde{\to}$ te$_{i_r}$,
100a act_fct$_j$: te$_{j_d}$ $\tilde{\to}$ te$_{j_r}$,

[32]http://en.wikipedia.org/wiki/Bézier_curve

```
100a        ... ,
100a        act_fct_k: $te_{k_d} \xrightarrow{\sim} te_{k_r}$
100b      where:
100b        type names in $(te_{(i|j|...|k)_d})$ and in $(te_{(i|j|...|k)_r})$ are
100b        type names defined by the indices which are prefixes of
100b        $\ell_m \frown \langle T_m \rangle$ and where $T_m$ is in some signature $act\_fct_{i|j|...|k}$.
```

5.2.3 Discovering Event Signature 274

Events are from the point of view of signatures very much like actions.

101. The 𝔼𝕍𝔼ℕ𝕋_𝕊𝕀𝔾ℕ𝔸𝕋𝕌ℝ𝔼𝕊 meta-function applies to an index set and yields
    a a set of action signatures each consisting of an action name and a pair of definition set and range type expressions where
    b the type names that occur in these type expressions are defined either in the domains indexed by the index set or by the environment (i.e., "outside" the domain Δ).

275

```
101     𝔼𝕍𝔼ℕ𝕋_𝕊𝕀𝔾ℕ𝔸𝕋𝕌ℝ𝔼𝕊: Index $\xrightarrow{\sim}$ **Text**
101     𝔼𝕍𝔼ℕ𝕋_𝕊𝕀𝔾ℕ𝔸𝕋𝕌ℝ𝔼𝕊($\{\ell_1\frown\langle T_1\rangle,\ell_2\frown\langle T_2\rangle,...,\ell_n\frown\langle T_n\rangle\}$):
101a        evt_fct_i: $te_{i_d} \xrightarrow{\sim} te_{i_r}$,
101a        evt_fct_j: $te_{j_d} \xrightarrow{\sim} te_{j_r}$,
101a        ... ,
101a        evt_fct_k: $te_{k_d} \xrightarrow{\sim} te_{k_r}$
101b      where:
101b        type names of $te_{(i|j|...|k)_d}$ and $te_{(i|j|...|k)_r}$ are type names
101b        defined by the indices which are prefixes of $\ell_m \frown \langle t_m \rangle$
101b        and where $t_m$ is in some signature $act\_fct_{i|j|...|k}$ or may
101b        refer to types definable only "outside" $\Delta$
```

5.2.4 Discovering Behaviour Signatures 276

We choose, in these research notes, to model behaviours in CSP[33]. This means that we model (synchronisation and) communication between behaviours by means of messages m of type M, CSP channels (**channel** ch:M) and CSP

output: ch!e [offer to deliver value of expression e on channel ch], and
input: ch? [offer to accept a value on channel ch].

[33]Other behaviour modelling languages are Petri Nets, MSCs: Message Sequence Charts, Statechart etc. We invite the reader to suggest corresponding 'discovery' techniques and tools.

We allow for the declaration of single channels as well as of one, two, ..., n dimensional arrays of channels with indexes ranging over channel index types

```
type  Idx, CIdx, RIdx ...:
channel ch:M, { ch_v[vi]:M′|vi:Idx }, { ch_m[ci,ri]:M″|ci:CIdx,ri:RIdx }, ...
```

etcetera. We assume some familiarity with RSL/CSP. 277

A behaviour usually involves two or more distinct sub-domains.

Example 57 (The Involved Subdomains of a Vehicle Behaviour) Let us illustrate that behaviours usually involve two or more distinct sub-domains. A vehicle behaviour, for example, involves the vehicle subdomain, the hub subdomain (as vehicles pass through hubs), the link subdomain (as vehicles pass along links) and, for the road pricing system, also the monitor subdomain. ●

278

102. The $\mathbb{BEHAVIOUR\_SIGNATURES}$ is a meta function.

103. It applies to a set of indices and results in a text,

104. The text contains

   a a set of zero, one or more message types,

   b a set of zero, one or more channel index types,

   c a set of zero, one or more channel declarations,

   d a set of one or more process signatures with each signature containing a behaviour name, an argument type expression, a result type expression, usually just **Unit**, and

   e an input/output clause which refers to channels over which the signatured behaviour may interact with its environment.

279

```
103.   BEHAVIOUR_SIGNATURES: Index $\xrightarrow{\sim}$ Text
103.   BEHAVIOUR_SIGNATURES($\{\ell_1{}^\frown\langle T_1\rangle,\ell_2{}^\frown\langle T_2\rangle,...,\ell_n{}^\frown\langle T_n\rangle\}$):
104a.          type    $M = M_1 \mid M_2 \mid ... \mid M_m$, $m\geq0$
104b.                  $I = I_1 \mid I_2 \mid ... \mid I_n$, $n\geq0$
104c.        channel ch,vch[i],{vch[i]:M|i:$I_a$},{mch[j,k]:M|j:$I_b$,k:$I_c$},...
104d.        value
104d.            $bhv_1$: $ate_1 \rightarrow \mathit{inout}_1$ $rte_1$,
104d.            $bhv_2$: $ate_2 \rightarrow \mathit{inout}_2$ $rte_2$,
104d.            ... ,
104d.            $bhv_m$: $ate_m \rightarrow \mathit{inout}_m$ $rte_m$,
104d.   where    type expressions $atei_i$ and $rte_i$ for all i involve at least two
104d.            types $t'_i$ and $t''_j$ of respective indexes $\ell_i{}^\frown\langle t_i\rangle$ and $\ell_j{}^\frown\langle t_j\rangle$
104e.   where    $\mathit{inout}_i$: in k | out k | in,out k
104e.   where    k: ch | ch[i] | {ch[i]|i ∈ $I_a$} | {mch[j,k]:M|i:$I_b$,j:$I_c$} | ...
```

280

Example 58 (A Vehicle Behaviour Signature Discovery) We refer, for example, to Examples 36 (Pages 56–61) and 40 (Pages 66–67).

```
let ih=⟨Δ.N.LS,H⟩,il=⟨Δ.N.HS,L⟩,iv=⟨Δ,F,V⟩,im=⟨Δ,Monitor⟩ in
BEHAVIOUR_SIGNATURES({iv,ih,iv,im}) as text
    let n:N, hs=obs_HS(n), ls=obs_LS(n), vs=obs_F(PART_SORTS)(⟨Δ⟩) in
    where text:
          type
             VL_Msg, VH_Msg, VM_Msg
          channel
    62a.     {vh_ch[ attr_VI(v),attr_HI(h) ]|v:V,h:H•v ∈ vd∧h ∈ hs}:VH_Msg
    62b.     {vl_ch[ attr_VI(v),attr_LI(h) ]|v:V,l:L•v ∈ vs∧h ∈ ls}:VL_Msg
    62c.     m_ch:VM_Msg
          value
    64.      vehicle: VI → V → VP →
    83.          out,in {vl_ch[ vi,li ]|li:LI•li ∈ xtr_LIs(ls)}
    83.                 {vh_ch[ vi,hi ]|hi:HI•hi ∈ xtr_HIs(hs)} out m_ch,... Unit
end end
```

•

5.3 What Does Application Mean ?

281

Now what does it actually mean "to apply" a discover function ? We repeat our list of discoverers.

[90 (Page 76)] PART_SORTS,

[91 (Page 77)] HAS_A_CONCRETE_TYPE,

[92 (Page 78)] PART_TYPES,

[93 (Page 79)] UNIQUE_ID,

[96 (Page 79)] MEREOLOGY,

[98 (Page 80)] ATTRIBUTES,

[100 (Page 82)] ACTION_SIGNATURES,

[101 (Page 83)] EVENT_SIGNATURES and

[103 (Page 84)] BEHAVIOUR_SIGNATURES.

It is the domain engineer cum scientist[34] who “issues” the “commands”. The first “formal” domain inquiry is that of $\mathbb{PART\_SORTS}(\langle\Delta\rangle)$. We refer to Item 90 on page 76, for example as captured by the formulas, Items 90–90b (Page 86). 282

For the domain engineer to ‘issue’ one of the ‘discovery commands’ means that that person has (i) prepared his mind to study the domain and is open to impressions, (ii) decided which $\mathcal{DISCOVERER\_KIND}$ to focus on, and (iii) studied the “rules of engagement” of that command, that is which pre-requisite discoverers must first have been applied, with which index, that is, in which context the command invocation should be placed, and which results the invocation is generally expected to yield.

5.3.1 PART_SORTS 283

Let us review the $\mathbb{PART\_SORTS}$ discoverer:

value
90. $\mathbb{PART\_SORTS}$: Index $\xrightarrow{\sim}$ **Text**
90. $\mathbb{PART\_SORTS}(\ell^\frown\langle T\rangle)$:
90a. tns:$\{T_1,T_2,...,T_m\}$:TN-**set** $\times$
90b. $\{$ obs_T_j: $T \to T_j \mid T_j$:tns$\}$

The domain analyser[35] has decided to “position” the search at domain index $\ell^\frown\langle \mathsf{T}\rangle$ where $\mathsf{T} = \Delta$ if $\ell = \langle\rangle$ and where T is some “previously discovered part type. 284

From Item 90a the domain analyser is guided (i.e., advised) to analyse the domain “at position $\ell^\frown\langle \mathsf{T}\rangle$: is the domain type T a composite type of one or more subpart types ? If so then decide which they are, that is: $\mathsf{T}_1,\mathsf{T}_2,...,\mathsf{T}_m$, that is, the “generation” of the text **type** $\mathsf{T}_1,\mathsf{T}_2,...,\mathsf{T}_m$, if not then $\mathsf{tns}=\{\}$ and no text is “generated”.

Item 90b, and given the domain analyser’s resolution of Item 90a, then directs the “generation” of m observers $\mathsf{obs\_T}_j$: $\mathsf{T} \to \mathsf{T}_j$ (for $j : \{1..m\}$).

5.3.2 HAS_A_CONCRETE_TYPE 285

Let us review the $\mathbb{HAS\_A\_CONCRETE\_TYPE}$ analyser:

91 $\mathbb{HAS\_A\_CONCRETE\_TYPE}$: Index $\to$ **Bool**
91 $\mathbb{HAS\_A\_CONCRETE\_TYPE}(\ell^\frown\langle T\rangle)$:**true**|**false**

Item 91 directs the domain analyser to decide whether the domain type T at “position” $\ell^\frown\langle t\rangle$ should be given a concrete type definition. It is a decision sôlely at the discretion of the domain analyser whether domain type T should be given a concrete type definition, and, as we shall see next, which concrete type it should then be “given”, that is, how it should be “concretely abstractly” modelled.

5.3.3 PART_TYPES 286

Let us review the $\mathbb{PART\_TYPES}$ analyser:

92. $\mathbb{PART\_TYPES}$: Index $\xrightarrow{\sim}$ **Text**
92. $\mathbb{PART\_TYPES}(\ell^\frown\langle t\rangle)$:
92a. **type** T = TE,

[34]When we write: *domain engineer cum scientist* we mean to say that the domain engineer really is performing a scientific inquiry.

[35]We use the alternative, synonymous terms: ‘domain engineer’, ‘domain describer’. ‘domain scientist’.

```
92b.            T_1 or T_1 = TE_1
92b.            T_2 or T_2 = TE_2
92b.            ...
92b.            T_n or T_n = TE_n
92c.       pre: HAS_A_CONCRETE_TYPE(ℓ⌢⟨t⟩)
```

The domain analyser has decided to "position" the search at domain index $\ell^\frown\langle\mathsf{T}\rangle$ where $\mathsf{T} = \Delta$ if $\ell = \langle\rangle$ and where T is some "previously discovered part type. 287

From Item 92 the domain analyser is guided (i.e., advised) to analyse the domain "at position $\ell^\frown\langle\mathsf{T}\rangle$: can a reasonably abstract, yet concrete type definion be given for T ? If so then decide which it should be, that is, should it be an atomic type a number type, **Intg, Rat, Real**, a Boolean type, **Bool**, or a token type, f.ex. TOKENA token type is a further undefined atomic type — typically used to model identifiers.; or should it be a composite type either a set type: TE: T_s**-set** of te: T_s**-infset**, or a Cartesian type: TE: $\mathsf{T}_1\times\mathsf{T}_2\times...\mathsf{T}_m$, or a list type: TE: $\mathsf{T}_\ell^\star$ or te: T_ℓ^ω or a map type: TE: $\mathsf{T}_d \xrightarrow[m]{} \mathsf{T}_t$? In either case the text type T = TE, T_1 or T_1=TE_1, T_2 or T_1=TE_2, ..., T_n or T_n=TE_n is generated where TE (TE_x) is a type expression whose so far undefined type names T_1, T_2, ..., T_n must be defined, either as sorts, or a concrete types. 288

5.3.4 UNIQUE_ID 289

Let us review the UNIQUE_ID analyser:

```
value
93.   UNIQUE_ID: Index → Text
93.a  UNIQUE_ID(ℓ⌢⟨T⟩):
93.b      type
93.c          TI
93.d      value
93.e          uid_TI: T → TI
```

Item 93.a inquires as to the Line 93.b **type** name Line 93.c of the inquired part type's unique identifiers Line 93.d and the function signature **value** Line 93.e of the observer, uid_TI, name, the definition set type (T, of course) and the range set type (TI — obviously). Thus, the only real "new" "discovery" here is the name, TI, of the unique identifier type. 290

Etcetera, etcetera.

5.4 Discussion

We have presented a set of discoverers:

[90 (Page 76)] PART_SORTS,

[91 (Page 77)] HAS_A_CONCRETE_TYPE,

[92 (Page 78)] PART_TYPES,

[93 (Page 79)] UNIQUE_ID,

[96 (Page 79)] MEREOLOGY,

[98 (Page 80)] ATTRIBUTES,

[100 (Page 82)] ACTION_SIGNATURES,

[101 (Page 83)] EVENT_SIGNATURES and

[103 (Page 84)] BEHAVIOUR_SIGNATURES.

291

There is much more to be said: About a meta-state component in which is kept the “text” so far generated. A component from which one can see which indices and hence which type names have so far been “generated”, and on the basis of which one can perform tests of well-formedness of generated text, etcetera, etcetera,

6 Conclusion 292

6.1 General

6.2 What Have We Achieved ? 293

6.3 Other Formal Models 294

6.4 Research Issues 295

6.5 Engineering Issues 296

6.6 Comparable Work 297

6.7 Acknowledgements 298