Domain Types[⋆]

Endurants

Dines Bjørner

Fredsvej 11, DK-2840 Holte, Danmark
DTU, DK-2800 Kgs. Lyngby, Denmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/˜dibj

1

[1]

2

Abstract. We put forward a new set of concepts. techniques and tools
for analysing and describing software application **domains**. We focus on
endurants leaving treatment of **perdurants** to another paper. Briefly, en-
durants are either **discrete** (and are called **parts**) or are **continuous** (and
are then called **materials**). Discrete parts are either **atomic**, that is, are 3
considered to not consist of proper subparts, or are **composite**. Parts
may be **mereologically** related: are uniquely identified and may be re-
lated to some other parts. Parts and materials can additionally be char- 4
acterised by **attributes**. Endurant attributes are here considered to be
non-manifest properties (other than unique identification and mereol-
ogy) which can be *observed*, but are otherwise intangible. The above 5
concepts give rise to a number of entity, endurant, perdurant, discrete-
ness, continuity, atomicity and composition **analysers**, and abstract part,
concrete part, unique identifier, mereology, and attribute **observers**.

1 Introduction 6

Before we can design software[2] (for some application) we must have a reasonable understanding of its requirements. Before we can prescribe requirements[3] (for that application) we must have a reasonable understanding of the domain of the requirements, that is, the domain in which the software is to be inserted.
So we see software development containing elements of domain description ($\mathcal{D}$), 7
requirements prescription ($\mathcal{R}$) and software design ($\mathcal{S}$) such that $\mathcal{D},\mathcal{S} \models \mathcal{R}$, that is we can prove correct implementation ($\mathcal{S}$) of requirements ($\mathcal{R}$) under assumptions of the domain ($\mathcal{D}$).

We shall focus on techniques and tools for analysing and describing domains. 8

• • •

⋆ Vertical grey margin bars designate changes with respect to an October 2013 version of this paper.

1 Margin numbers coordinate with slide page numbers of a lecture slides version of this paper.

2 Software includes the code that executes on a computer.

3 Requirements: the properties that the software must fulfill.

Describing domains prior to requirements prescription is like researching physics before engineering. Nobody in their right mind would employ space rocket engineers who are not well educated in fluid mechanics – or radio transmission engineers who are not well educated in Maxwell's Equations.

2 Endurants 9

2.1 Domains

A **domain** is characterised by its observable, i.e., manifest *entities* and their *qualities* ● [4]

Example 1 Domains. *a canal complex, a road net (a canal complex modulo materials), a container line, a pipeline, a hospital*■ [5]

2.2 Sorts, Types and Domain Analysis 10

By a **sort** (or **type** – which we take to be the same) we shall understand the largest set of entities all of which have the same qualities[6] ● 11

Example 2 Sorts. *Links*[7] *of any road net constitute a sort. So does hubs*[8]*. The largest set of (well-formed) collections of links constitute a sort. So does similar collections of hubs. The largest set of road nets (containing well-formed collections of hubs and links) form a sort*■

12 By **domain analysis** we shall understand a process whereby a domain analyser groups entities of a domain into sorts ● The rest of this paper will outline a class of domain analysis principles, techniques and tools — focusing on endurant entities.

2.3 Entities and Qualities 13

By an **entity** we shall understand a phenomenon that can be observed, i.e., be seen or touched[9] by humans, or that can be conceived as an abstraction of an entity[10] ● 14 The method can thus be said to provide the *domain analysis prompt*: `is_entity` where `is_entity(`θ`)` holds if θ is an entity.

Example 3 Entities. *(a) a road net, (b) a link of a road net, (c) a hub of a road net;* and *(d) insertion of a link in a road net, (e) disappearance of a link of a road net, and (f) the movement of a vehicle on a road net*■

[4] **Definitions** start with a single quoted **'term'** and conclude with a ●
[5] **Examples** conclude with a ■
[6] Taking a sort (type) to be *the largest set of entities all of which have the same qualities* reflects Ganter & Wille's notion of a **formal concept** [26].
[7] A link: a street segment between two adjacent hubs
[8] A hub: an intersection of street segments
[9] An entity which can be seen or touched is thus a physical phenomenon. If an entity has the quality the colour red, it is not the red that is an entity.
[10] There is no "infinite loop" here: a concept can be an abstraction of (another) concept, etc., which is finally an abstraction of a physical phenomenon.

Qualities. By a **quality** of an entity we shall understand a property that can be given a name and whose value can be precisely measured by physical instruments or otherwise identified • 15 16

Example 4 Quality Names. *Name, Owner, Length, Location, Cadestral Hub Location, Hub State, Hub State Space*[11], etcetera ■

Example 5 Quality Values. *the name of a road net, the ownership of a road net, the length of a link, the location of a hub*, etcetera ■

2.4 Endurants and Perdurants 17

Entities are either endurants or are perdurants.

By an **endurant entity** (or just, an endurant) we shall understand anything that can be observed or conceived, as a "complete thing", at no matter which given snapshot of time. Were we to "freeze" time we would still be able to observe the entire endurant • Thus the method provides a *domain analysis prompt*: 18
`is_endurant` where `is_endurant`(e) holds if entity e is an endurant.

Example 6 Endurants. Items (a–b–c) of Example 3 on the facing page are endurants; so are the *pipes, valves,* and *pumps* of a pipeline.

Perdurants. By a **perdurant entity** (or just, an perdurant) we shall understand an entity for which only a fragment exists if we look at or touch them at any given snapshot in time, that is, were we to freeze time we would only see or touch a fragment of the perdurant • Thus the method provides a *domain analysis prompt*: 19 20
`is_perdurant` where `is_perdurant`(e) holds if entity e is a perdurant.

Example 7 Perdurants. Items (d–e–f) of Example 3 on the preceding page are perdurants; so are the *insertion of a hub, removal of a link,* etcetera ■

2.5 Discrete and Continuous Endurants 21

Entities are either discrete or are continuous.

Discrete Endurants. By a **discrete endurant** we shall understand something which is separate or distinct in form or concept, consisting of distinct or separate parts • We use the term **part** for discrete endurants, that is: `is_part`$(p)\equiv$ `is_endurant`$(p)\wedge$`is_discrete`(p) • Thus the method provides a *domain analysis prompt*: `is_discrete` where `is_discrete`(e) holds if entity e is discrete. 22

Example 8 Discrete Endurants. The examples of Example 6 are all discrete endurants ■

Continuous Endurants. By a **continuous endurant** we shall understand something which is prolonged without interruption, in an unbroken series or pattern ● 23
We use the term **material** for continuous endurants ● Thus the method provides a *domain analysis prompt*: `is_continuous` where `is_continuous`(e) holds if entity e is continuous. 24

Example 9 Continuous Endurants. The pipes, valves, pumps, etc., of Example 6 on the previous page may contain *oil; water* of a hydro electric power plant is also a material (i.e., a continuous endurant) ■

● ● ●

We are not covering perdurants in this paper.

2.6 Parts and Materials 25

Atomic and Composite Discrete Endurants. Discrete endurants are either atomic or are composite.

26 **Atomic Endurants.** By an **atomic endurant** we shall understand a discrete endurant which in a given context, is deemed to *not* consist of meaningful, separately observable proper sub-parts ● The method can thus be said to provide the *domain analysis prompt*: `is_atomic` where `is_atomic`(p) holds if p is an
27 atomic part.

Example 10 Atomic Parts. Examples of atomic parts of the above mentioned domains are: *aircraft (of air traffic), demand/deposit accounts (of banks), containers (of container lines), documents (of document systems), hubs, links and vehicles (of road traffic), patients, medical staff and beds (of hospitals), pipes, valves and pumps (of pipeline systems), and rail units and locomotives (of railway systems)* ■

28 **Composite Endurants.** By a **composite endurant** we shall understand a discrete endurant which in a given context, is deemed to *indeed* consist of meaningful, separately observable proper sub-parts ● The method can thus be said to provide the *domain analysis prompt*: `is_composite` where `is_composite`(p) holds if p is
29 an a composite part.

Example 11 Composite Parts. Examples of composite parts of the above mentioned domains are: *airports and air lanes (of air traffic), banks (of a financial service industry), container vessels (of container lines), dossiers of documents (of document systems), routes (of road nets), medical wards (of hospitals), pipelines (of pipeline systems), and trains, rail lines and train stations (of railway systems)* ■

30 It is the domain analysers who decide whether an endurant is atomic or composite. In the context of air traffic an aircraft might very well be described as an atomic entity; whereas in the context of an airline an aircraft might very well be described as a composite entity consisting of the aircraft 'body', the crew, the passengers, their luggage, the fuel, etc.

Part Observers. From atomic parts we cannot observe any sub-parts. But from composite parts we can. 31

For composite parts, p, the *domain description prompt*

`observe_part_sorts`(p)

yields some *formal description text* according to the following *schema*[12]:

type P_1, P_2, ..., P_n;[13]
value **obs_**P_1: $\mathsf{P}\to\mathsf{P}_1$, **obs_**$\mathsf{P}_2$: $\mathsf{P}\to\mathsf{P}_2$,...,**obs_**$\mathsf{P}_n$: $\mathsf{P}\to\mathsf{P}_n$;[14]

where sort names P_1, P_2, ..., P_n are chosen by the domain analyser, must denote disjoint sorts, and may have been defined already, but not recursively A proof obligation may need be discharged to secure disjointness. 32

Example 12 Abstract Canal Complexes.

1. *A canal complex consists of a cluster of canal links and a cluster of canal hubs.*

type
1. *CC, CH, CL*
value
1. *obs_CH: CC → CH, obs_CL: CC → CL*

This example will be continued in Examples 13, 14, 15, etcetera.

Sort Models. A part sort is an abstract type. Some part sorts, P, may have a concrete type model, T. Here we consider only two such models: one model is as sets of parts of sort A: T = A-**set**; the other model has parts being of either of two or more alternative, disjoint sorts: T=P1|P2|...|PN. The *domain analysis prompt*: 33

`has_concrete_type`(p)

holds if part p has a concrete type. In this case the *domain description prompt*

`observe_concrete_type`(p)

yields some *formal description text* according to the following *schema*, 34

- either

  type P1, P2, ..., PN, T = $\mathcal{E}$(P1,P2,...,PN)[15]
  value **obs_**T: P → T[16]

  where $\mathcal{E}$(...) is some type expression over part sorts and where P1,P2,...,PN are either (new) part sorts or are auxiliary (abstract or concrete) types[17];
- or:

  type
  T = P1 | P2 | ... | PN[18]
  P_1, P_2, ..., P_n
  P1 :: mkP1(P_1), P2 :: mkP2(P_2), ..., PN :: mkPN(P) [19]
  value
  obs_T: P → T[20]

35

Example 13 **Concrete Canal Clusters**. *We continue Example 12.*

2. *A cluster of canal links is a set of links.*
3. *A cluster of canal hubs is a set of hubs.*

type
*2-3. L, H, CLS = L-***set***, CHS = H-***set**
value
2-3. obs_CLS = CL→CLS, obs_CHS = CH→CHS

36 **Material Observers.** Some parts p of sort P may contain material. The *domain analysis prompt*

has_material(p)

[11] with values being the set of all hub states over time.
[12] Throughout this paper the description texts are formulated in the RAISE [28] specification language RSL [27] – but other such model-oriented specification languages could be used, e.g., Alloy [30], Event B [1], VDM [18, 19, 23], or Z [40].
[13] This RSL **type** clause defines P_1, P_2, ..., P_n to be types.
[14] Thus RSL **value** clause defines n function values. All from type P into some type P_i.
[15] The concrete type definition T = $\mathcal{E}$(P1,P2,...,PN) define type T to be the set of elements of the type expressed by type expression $\mathcal{E}$(P1,P2,...,PN).
[16] **obs_**T is a function from any element of P to some element of T.
[17] The *domain analysis prompt*: sorts_of(t) yields a subset of {P1,P2,...,PN}.
[18] A|B is the union type of types A and B.
[19] Type definition A :: mkA(B) defines type A to be the set of elements mkA(b) where b is any element of type B
[20] **obs_**T is a function from any element of P to some element of T.

holds if composite part p contains one or more materials. The *domain description prompt*

`observe_material_sorts(`p`)`

yields some *formal description text* according to the following *schema*:

type M_1, M_2, ..., M_m;
value **obs_**M_1: P → M_1, **obs_**M_2: P → M_2, ..., **obs_**M_m: P → M_m;

where values, m_i, of type M_i satisfy `is_material`(m) for all i; and where M_1, M_2, ..., M_m must be disjoint sorts. 37

Example 14 Canal Complex Materials.

4. *Canal links and hubs contain (more or less) water.*
5. *$\mathcal{W}_\emptyset$ designates no water.*

type
4. *W*
value
4. *obs_W: (L|H) → W*
5. *$\mathcal{W}_\emptyset$:W*

38
Some material m of sort M may contain parts. The *domain analysis prompt*

`has_parts(`m`)`

holds if material m contains one or more parts. The *domain description prompt*

`observe_part_sorts(`m`)`

yields some *formal description text* according to the following *schema*:

type P_1, P_2, ..., P_n;
value **obs_**P_1: M→P_1, **obs_**P_2: M→P_2,...,**obs_**P_m: M→P_m;

where values, p_i, of type P_i satisfy `is_part`(p_i) for all i; and where sort expressions P_1, P_2, ..., P_n must designate disjoint sorts. 39

Example 15 Canal Boats.

6. *Canal water (also $\mathcal{W}_\emptyset$) contains zero or more boats.*

type
6. *B*
value
6. *obs_Bs: W → B-**set***

2.7 Endurant Properties 40

We have already, above, treated the following properties of endurants: `is_discrete`, `is_continuous`, `is_atomic`, `is_composite` and `has_material`. We may think of those properties as external qualities. In contrast we may consider the following internal qualities: `has_unique_identifier` (parts), `has_mereology` (parts) and `has_attributes` (parts and materials).

2.8 Unique Identifiers 41

Without loss of generality we can assume that every part has a unique identifier[21]. A **unique part identifier** (or just unique identifier) is a further undefined, abstract quantity. If two parts are claimed to have the same unique identifier then they are identical, that is, their possible mereology and attributes are (also) identical ● The *domain description prompt*:

`observe_unique_identifier`(p)

yields some *formal description text* according to the following *schema*:

type PI;
value **uid**_P: P → PI;

42

Example 16 Unique Identifiers. *A road net consists of a set of hubs and a set of links. Hubs and links have unique identifiers. That is:* **type** **HI, LI;** **value** **uid_H: H→HI, uid_L: L→LI;** ■

2.9 Mereology 43

By **mereology** [33] we shall understand the study, knowledge and practice of parts, their relations to other parts and "the whole" ●

Part relations are such as: two or more parts being connected, one part being embedded within another part, and two or more parts sharing (other) attributes.

44

Example 17 Mereology. *The mereology of a link of a road net is the set of the two unique identifiers of exactly two hubs to which the link is connected. The mereology of a hub of a road net is the set of zero or more unique identifiers of the links to which the hub is connected*■

45

The *domain analysis prompt*: `has_mereology`(p) holds if the part p is related to some others parts $(p_a, p_b, \ldots, p_c)$. The *domain description prompt*:

`observe_mereology`(p)

can then be invoked and yields some *formal description text* according to the following *schema*:

[21] That is, `has_unique_identifier`(p) for all parts p.

type MT = $\mathcal{E}$(PI$_A$,PI$_B$,...,PI$_C$);
value **mereo_P**: P → MT;

where $\mathcal{E}$(...) is some type expression over unique identifier types of one or more part sorts. Mereologies are expressed in terms of structures of unique part identifiers. Usually mereologies are constrained. Constraints express that a mereology's unique part identifiers must indeed reference existing parts, but also that these mereology identifiers "define" a proper structuring of parts. 46 47

Example 18 Mereology Constraints. *Canal Complexes:*

7. *A link connects exactly two distinct hubs, and this is expressed by associating with each link a set of the two distinct hub idenfiers of the hubs they connect.*
8. *A hub connects to a set of links, and this is expressed by associating with a hub set set of the zero, one or more identifiers of the links to which it connects.*

48

```
value
ι7. mereo_L: L → HI-set axiom ∀ l:L•card mereo_L(l)=2
ι8. mereo_H: H → LI-set
```

49

9. The unique hub identifiers of a link mereology must designate existing hubs, and
10. the unique link identifiers of a hub mereology must designate existing links.

```
axiom
ι9.   ∀ cc:CC,ch:CH,cl:CL,hs:Chs,ls:CLs •
ι9.      ch=obs_CH(cc)∧cl=obs_CL(cc)∧hs=obs_CHs(ch)∧ls=obs_CLs(cl)⇒
ι9.         ∀ l:L•l ∈ ls ⇒
ι9.            ∃ h,h′:H • {h,h}⊆hs∧mereo_L(l)={uid_H(h),uid_H(h′)}
ι10.        ∧ ∀ h:H•h ∈ hs ⇒
ι10.           ∀ li:LI•li ∈ mereo_H(h) • ∃ l:L•l ∈ ls∧uid_L(l)=li
```
■

50

Two parts, p_i:P_i and p_j:P_j, of possibly the same sort (i.e., P_i≡P_j) are said to **refer one to another** if the mereology of p_i contains the unique identifier of p_j and vice-versa • The parts p_i and p_j are then said to enjoy **part overlap** • We refer to the concept of shared attributes covered at the very end of this section.

2.10 Attributes 51

Attributes are what really endows parts with qualities. The external properties[22] are far from enough to distinguish one sort of parts from another. Similarly with unique identifiers and the mereology of parts. We therefore assume, without loss of generality, that every part, whether discrete or continuous, whether, when discrete, atomic or composite, has at least one attribute. 52

[22] `is_discrete`,`is_continuous`,`is_atomic`,`is_compositehas_material`.

By an **emdurant attribute**, we shall understand a property that is associated with an endurant e of sort E, and if removed from endurant e, that endurant would no longer be endurant e (but may be an endurant of some other sort E'); and where that property itself has no physical extent (i.e., volume), as the endurant may have, but may be measurable by physical means • The *domain description prompt* 53

observe_attributes(p)

yields some *formal description text* according to the following *schema*:

type A_1, A_2, ..., A_n, ATTR;
value **attr_**A_1:P→A_1, **attr_**A_2:P→A_2, ..., **attr_**A_n:P→A_n,
  attr_ATTR:P→ATTR;

where **for** ∀ p:P, **attr_**A_i(**attr_**ATTR(p)) ≡ **attr_**A_i(p). 54

Example 19 Canal Complex Attributes.

11. *Links are endowed with attributes such as lengths and widths, locations, names, etc., and hubs can likewise be endowed with properties such as locations, names, etc.*
12. *Hubs are endowed with the additional attribute of 'hub state'. A hub state models the directions that traffic may, in time intervals, proceed through a hub. We abstract a hub state as a set of pairs of link identifiers.*

type
11. LEN, WID, LOC, NAM, ...
value
11. attr_LEN: L→LEN, attr_WID: L→WID,
attr_LOC: (L|H)→LOC, attr_NAM: (L|H)→NAM, ...
type
*12. HΣ = (LI×LI)***-set**
value
12. attr_HΣ: H → HΣ

55

13. Directional flow is a concept of unit volume of water per unit time. Flows are directionaless directional flows. Flows are observed at the interfaces between links and hubs (from a link to a hub or vice versa), and at the interfaces between links or hubs and a further undefined exterior.
14. An interface is either from a hub to a link (hlif, or reverse: from a link to a hub[23], lhif) or between a hub or link and an outside.
15. Water may (directionally) flow across a hub/link (or a link/hub) interface (in various and varying quantities).

[23] The reverse direction of an interface **designates** the same interface with opposite direction flows (for non-null flows); see axiom ι18 (Page 11).

16. Water may *leak* from links and hubs (evaporate into the air[24] or “sink” into the ground[25]),
17. A nd water may *precipitate* from the air into hubs and links, or otherwise enter these from possibly underground sources.

56

```
type
ι13.  F
value
      $\mathcal{O}_{\mathcal{F}}$: F
      −: F → F,  +,−: F×F→F
axiom
      ∀ f:F•$\mathcal{O}_{\mathcal{F}}$+f=f+$\mathcal{O}_{\mathcal{F}}$=f=$\mathcal{O}_{\mathcal{F}}$−f=f−$\mathcal{O}_{\mathcal{F}}$
type
ι14.  IF == hlif(hi:HI,li:LI) | lhif(li:LI,hi:HI)
value
ι15.  attr_F: IF → CC → F
ι16.  attr_Leak: (Li|Hi) → CC → F
ι17.  attr_Prec: (Li|Hi) → CC → F
```

57

Laws of Material Flow: We shall focus on laws of stead state flows, that is, flows which for which the volumes of water in links and hubs are unchanged.

18. The flow across a link to hub interface is the numerically the same as the flow across the “same”[26], but now reversed link to hub interface,

```
ι18.  ∀ cc:CC,cl:CL,ls:CLs • cl=obs_CL(cc)∧ls=obs_CLs(cl)⇒
ι18.      ∀ l:L•l ∈ ls ⇒
ι18.          let li = uid_L(l),
ι18.          ∀ hi:HI • hi ∈ mereo_L(l)
ι18.              ⇒ attr_F(hlif(hi,li))(cc) = − attr_F(lhif(li,hi))(cc)
ι18.      end
```

58

19. The flow of water from a hub into a designated, that is, a connected link plus the precipitation along the link equals the flow of water from that link into “the other connected” hub plus the leak from that link.

```
axiom
ι19.  ∀ cc:CC,cl:CL,ls:CLs • cl=obs_CL(cc)∧ls=obs_CLs(cl)⇒
ι19.      ∀ l:L•l ∈ ls ⇒
ι19.          let li = uid_L(l), {hi,hi′} = mereo_L(l) in
ι19.          attr_F(hlif(hi,li))(cc) + attr_Prec(li)(cc)
ι19.          = attr_F(lhif(li,hi′))(cc) + attr_Leak(li)(cc)
ι19.      end
```

59

20. The flow of water into a hub from (all) its connected links plus the precipitation at the hub equals the flow of water from the hub into (all) its connected links plus the leak from that hub.
21. The flow into and out of a hub is expressed by the summation function sum_F.

```
axiom
ι20.  ∀ cc:CC,ch:CH,hs:Chs • ch=obs_CH(cc)∧hs=obs_CHs(ch)⇒
ι20.      ∀ h:H•h ∈ hs ⇒
ι20.          let hi = uid_H(h), lis = mereo_H(h) in
ι20.          sum_F(lis,hi)(cc) + attr_Prec(li)(cc) = attr_Leak(hi)(cc)
ι20.      end
value
ι21.  sum_F: LI-set × HI → CC → F
ι21.  sum_F({},hi)(cc) = 𝒪_ℱ
ι21.  sum_F({li}∪ lis,hi)(cc) = attr_F(li,hi)(cc) ∪ sum_F(lis,hi)(cc)
```

60 **Shared Attributes.** A final quality of endurant entities is that they may share attributes. Two parts, p_i:P_i, p_j:P_j, of different sorts are said to enjoy **shared attributes** if P_i and P_j have at least one attribute name in common ● In such cases the mereologies of p_i and p_j are expected to refer to one another, i.e., be
61 **commensurable**.

Example 20 Shared Attributes. *Assume a road net, i.e., a canal complex modulo water. Assume that the road net domain (besides hub and link clusters) also include fleets of busses and a global bus monitoring part which embodies a global bus timetable as an attribute. Each bus is supposed to run according to that global bus timetable — which we assume that the bus has a copy of – as a bus attribute. Then the global bus monitoring part and the bus timetables of the bus parts of the fleet part are shared*■

62 **Attribute Categories.** One can suggest a hierarchy of endurant attribute categories: discrete or continuous (including chaotic) values, static or dynamic values (and within the dynamic value category: inert values or reactive values or active values (and within the dynamic active value category: autonomous values or biddable values or programmable values)).

We now review[27] these attribute value types. (The review is inspired by [31,
63 M.A. Jackson].)

[24] We do not model the 'air'.
[25] We do not model the 'ground'.
[26] See footnote 23 on Page 10.
[27] The margin bar stretches this review. The review can be omitted in order to comply with a 15 page maximum for FM 2014 contributions.

Attributes are either **discrete** or **continuous** or **chaotic** attributes. An attribute is said to be a **discrete attribute**, `is_discrete_attribute`, if it takes on a finite or countably infinite number of distinct or unconnected elements. An attribute is said to be a **continuous attribute**, `is_continuous_attribute`, if a suitable abstract model describes it as a continuous function from some point set value type A (A could be time) to some not necessarily point set value type (B): A → B. We shall not explain concepts of chaotic attributes. 64

Discrete or continuous part attributes are either constant or a variable, i.e., **static** or **dynamic** attributes. By a **static attribute** `is_static_attribute`, we shall understand an attribute whose values are constants, i.e., cannot change. By a **dynamic attribute**, `is_dynamic_attribute`, we shall understand an attribute whose values are variable, i.e., can change. 65

Dynamic attributes are either inert, reactive or active attributes. By an **inert attribute**, `is_inert_attribute`, we shall understand a dynamic attribute whose values only change as the result of external stimuli where these stimuli prescribe exactly these new values. By a **reactive attribute**, `is_reactive_attribute`, we shall understand a dynamic attribute whose values, if they vary, change value in response to the change of other attribute values. By an **active attribute**, `is_active_attribute`, we shall understand a dynamic attribute whose values change (also) of its own volition. 66

Example 21 Inert and Reactive Attributes. *Busses (i.e., vehicles) have a timetable attribute which is dynamic, i.e., can change, namely when the operator of the bus fleet decides so, thus the bus timetable attribute is inert. Pipeline valve units include the two attributes of valve opening (`open, close`) and internal flow (measured, say `gallons` per `second`). The valve opening attribute is of the programmable attribute category. The flow attribute is reactive (flow changes with valve opening/closing).*

67

Active attributes are either autonomous, biddable or programmable attributes. By an **autonomous attribute**, `is_autonomous_attribute`, we shall understand a dynamic active attribute whose values change value only "on their own volition". The values of an autonomous attributes are a "law onto themselves and their surroundings". By a **biddable attribute**, `is_biddable_attribute` (of a part) we shall understand a dynamic active attribute whose values may be subject to a contract as to which values it is expected to exhibit. By a **programmable attribute**, `is_programmable_attribute`. we shall understand a dynamic active attribute whose values can be accurately prescribed. 68

Example 22 Static and Dynamic Link Attributes.

22. Some link attributes
   a (link) length,
   b (link) name, e.g., Fifth Ave. between 50th and 51st Streets),
can be considered static,
23. whereas other link attributes
   a (link) state (one-way: say from 51st to 50th),

b (link) state space (one single state, one-way: say from 51st to 50th)
can be considered programmable,

24. Finally link attributes
a link state–of–repair,
b date last maintained,
can be considered inert.
69

```
type
22a.    LEN
value
22a.    obs_LEN: L → LEN
type
22b.    Name
value
22b.    obs_Name: L → Name
type
23a.    LΣ′=(HI×HI)-set
23a.    LΣ={|lσ:LΣ • card lσ ≤ 2|}
value
23a.    obs_LΣ: L → LΣ
type
23b.    LΩ′=LΣ-set
23b.    LΩ={|lω:LΩ • card lω = 1|}
value
23b.    obs_LΩ: L → LΩ
type
24a.    LSoR
24b.    DLM
value
24a.    obs_LSoR: L → LSoR
24b.    obs_DLM: L → DLM
```
70

Example 23 Autonomous and Programmable Hub Attributes. *We continue Example 22. Time progresses autonomously, Hub states are programmed (traffic signals): changing from red to green via yellow, in one pair of (co-linear) directions, while changing, in the same time interval, from green via yellow to red in the "perpendicular" directions.*

• • •

71 The attribute categories must not be confused neither with the external endurant properties nor with the endurant attributes.

2.11 Summary 72

Endurant sort names denote a usually infinite set of entities. Endurant mereology, when discrete, and attributes — for either discrete or continuous endurants — designate entity qualities.

We usually do not define endurant entities, that is, instances of endurants, but their sorts, mereology types and attribute types.

3 Perdurants 73

We give a very cursory summary of perdurant entities: actions, events and behaviours; their signatures (i.e., names and types) and qualities. 74

Perdurants, although of temporary nature, usually leave an effect. The effect
is that of changing a state. One can consider any collection of “disjoint” en-
durants that each contain dynamic attributes to form a **state**. That is, the state
is then represented by the possibly varying values of the dynamic attributes
of state endurants ● Perdurants are, for pragmatic purposes, classified as either 75
actions, events, or behaviours. An **action** is what happens when a function is 76
deliberately applied to some arguments and a state ● An **event** is what happens
when a function with a non-deterministic outcome is surreptitiously “applied”
to a state ● A **behaviour** is a sequence of zero or more actions, events and be-
haviours ● Since perdurants (potentially) change some notion of state we shall 77
basically ascribe functional values to perdurants – with these values charac-
terised by type and properties. Perdurant types are basically over states, that
is, $\Sigma \rightarrow \Sigma$, and with arguments that are either parts, materials or attributes,
or designate parts (i.e., their unique identifiers). Thus part, material, attribute
and unique identifier types occur in the function signatures. 78

● ● ●

We usually do not define perdurant entities, that is, instances of perdurants,
like specific actions, specific events or specific behaviours, but we define their
signatures: names and types and their qualities. Usually endurant qualities are
given by defining pre/post conditions. Each perdurant definition then usually
denotes an infinity of perdurants: actions, events or behaviours, 79

We can postulate a **normal form** behavioral CSP [29] description of a domain as follows: With each part, p:P, we associate a CSP process: **value** P: PI × ATTRIBS[28] → **in** ... **out** ... **Unit**. And with each unique identifier, π', in the mereology of a part p:P (whose unique identifier is π) we associate a CSP **channel** ch[$\{\pi, pi'\}$] between processes p:P and p':P'. Details are given in [10].

4 Conclusion 80

It is time to conclude.

4.1 What Have We Achieved ?

We have put forward a proposal for analysing and describing endurant entities of
domains. The proposal hinges on the notions of atomic and composite parts and
of materials, and on their analysis and description in terms of their sorts (**type**s)
and **obs**ervers. Based on parts and materials the proposal then proceeds with
the notions of mereology, including unique identifiers of parts, and with attributes
of parts and materials, and their observer **uid**, **mereo** and **attr** names and
their **type**s. We claim that this approach to the analysis and description of 81
endurants is new. In [14, Sect. 4.1] (See Sect. 4.3 below) we substantiate this
claim. In other words: the present paper presents an altogether different set of

[28] ATTRIBS are the attributes of P.

endurant concepts and thus an altogether different approach to the analysis and description of endurants, (parts, materials, unique identification, mereology, and attributes) than heretofore published.

4.2 On Relations to Philosophy 82

Already the very first (definitional) phrase of Sect. 2.3 on Page 2 reveals a problem. The definition is not mathematically precise. It cannot be made so. This is so since it we are dealing with reality. And reality, we take as a dogma, cannot be described precisely. The philosophical papers [38, 39] addresses this
83 dilemma,

But it is not only the definition of entity (Sect. 2.3 on Page 2) which is necessarily imprecise. So are the definitions of quality (Sect. 2.3 on Page 3), endurant (Sect. 2.4 on Page 3), perdurant (Sect. 2.4 on Page 3), discrete endurant (Sect. 2.5 on Page 3), continuous endurant (Sect. 2.5 on Page 4), atomic endurant
84 (Sect. 2.6 on Page 4), composite endurant (Sect. 2.6 on Page 4), etcetera.

But it is not only aspects of parts, atomicity and compositionality whose "reality" orders strongly on philosophical issues, so does issues of part-hood, that is mereology [20]. and attributes. The analysis and description of attributes
85 is subject to much philosophical debate and discussion as witnessed by [35, 25].

In other words: Describing domains can never be a precise activity. It depends on the pragmatics of the analyser cum describer. But with the given observers (**obs_**· · ·, **uid_**· · ·, **mereo_**· · ·, and **attr_**· · ·) the descriptions become precise!

We can therefore postulate that the domain analysis and description method of this paper sum up what can be described!

4.3 Comparison to Other Works 86

[14] contains a detailed, 4+ page, comparison to seminal works relating to domain analysis. We do not repeat this comparison here other than give references to the most important of this 'other work': [2, 22, 3, 21, 24, 32, 34, 36, 37]. .

4.4 Future Work 87

[13, Draft] proposes an approach to the analysis & description of domain perdurants that is commensurate with the ideas of the present paper.

Other ongoing work relates to the modelling of the analysis and description process. [14, April 2014] formalises some of the aspects of the process presented
88 in this paper. [11, Draft] formalises remaining process aspects.

The state-of-the-art with respect to the modelling of of combined continuous and discrete systems is somewhat deficient. A case in point is that of the dynamic flows of canal complexes. Examples 12, 14, 18 and 19 suggest models of the dynamic flow of water through a canal complex — for example "integrating" Bernoulli and Navier-Stokes equations with the discrete `RSL` specifications. We pose this kind of modelling as a challenge.

4.5 Domain Description Sketches 89

A number of sketches of domains based on the principles of this paper have been carried out. Some are: container lines (℧/container-paper), platoons & platooning (℧/platoon-p), road transport (℧/road-p), pipelines (℧/pipe-p), documents (℧/doc-p), window and web-based systems: transaction processing (℧/wf-dftp), stock exchanges (℧/tse-1), the market: consumers, retailers, wholesaler, producers (℧/themarket), logistics (℧/logistics), and licensing (℧/license--languages) — where ℧/file abbreviates www.imm.dtu.dk/~db/file.pdf.

4.6 Acknowledgements 90

I thank the very many colleagues around the world who have kindly invited me to lecture on domain science and engineering (domain analysis & description) in recent years:Jin Song Dong, Dominique Méry, Franz Wotawa, Jimmy Ho Man Lee, Wolfgang J. Paul, Alan Bundy, Tetsuo Tamai, Jens Knoop, Lars-Henrik Ericsson, Magne Haveraaen, Sun Meng and Zhu Hui Biao. The present form of domain analysis and description by means of prompts was spurred by some remarks of Magne Haveraaen.

5 Bibliography 91

5.1 Notes

This conference contribution is part of a series of papers on the topic of domains. [4–10, 12, 14–17]. In [5, 2008] we show how to "derive" requirements prescriptions from domain descriptions. In [6, 2008] we show techniques for describing domain facets: intrinsics, support technologies, rules & regulations, management & organisation as well as human behaviour. In [9, 2011] we illuminate such concepts as simulation, demos, monitoring and control. In [12, 2013] we speculate on various issues of "computation for humanity" (!). In [10, 2013] we relate our modelling of mereology to the classical axiom systems for mereology.

5.2 References

1. J.-R. Abrial. The B Book: Assigning Programs to Meanings *and* Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge, England, 1996 and 2009.
2. C. Bachman. Data structure diagrams. *Data Base, Journal of ACM SIGBDP*, 1(2), 1969.
3. V. Benjamins and D. Fensel. The Ontological Engineering Initiative (KA)2. Internet publication + Formal Ontology in Information Systems, University of Amsterdam, SWI, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands and University of Karlsruhe, AIFB, 76128 Karlsruhe, Germany, 1998. http://www.aifb.uni-karlsruhe.de/WBS/broker/KA2.htm.

4. D. Bjørner. Domain Theory: Practice and Theories, Discussion of Possible Research Topics. In *ICTAC'2007*, volume 4701 of *Lecture Notes in Computer Science (eds. J.C.P. Woodcock et al.)*, pages 1–17, Heidelberg, September 2007. Springer.
5. D. Bjørner. From Domains to Requirements. In *Montanari Festschrift*, volume 5065 of *Lecture Notes in Computer Science (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer)*, pages 1–30, Heidelberg, May 2008. Springer.
6. D. Bjørner. Domain Engineering. In P. Boca and J. Bowen, editors, *Formal Methods: State of the Art and New Directions*, Eds. Paul Boca and Jonathan Bowen, pages 1–42, London, UK, 2010. Springer.
7. D. Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics, Part I of II: The Engineering Part*. *Kibernetika i sistemny analiz*, (4):100–116, May 2010.
8. D. Bjørner. Domain Science & Engineering – *From Computer Science to The Sciences of Informatics Part II of II: The Science Part*. *Kibernetika i sistemny analiz*, (2):100–120, May 2011.
9. D. Bjørner. Domains: Their Simulation, Monitoring and Control – A Divertimento of Ideas and Suggestions. In *Rainbow of Computer Science, Festschrift for Hermann Maurer on the Occasion of His 70th Anniversary.*, Festschrift (eds. C. Calude, G. Rozenberg and A. Saloma), pages 167–183. Springer, Heidelberg, Germany, January 2011.
10. D. Bjørner. *A Rôle for Mereology in Domain Science and Engineering*. Synthese Library (eds. Claudio Calosi and Pierluigi Graziani). Springer, Amsterdam, The Netherlands, October 2013.
11. D. Bjørner. Domain Analysis: A Model of Prompts (paper[29], slides[30]). Research Report 2013-6, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Fall 2013.
12. D. Bjørner. *Domain Science and Engineering as a Foundation for Computation for Humanity*, chapter 7, pages 159–177. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC [Francis & Taylor], 2013. (eds.: Justyna Zander and Pieter J. Mosterman).
13. D. Bjørner. Domain Analysis & Description: Perdurants. Research Report, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Fall/Winter 2014.
14. D. Bjørner. Domain Analysis: Endurants – An Analysis & Description Process Model. In S. Iida, J. Meseguer, and K. Ogata, editors, *Specification, Algebra, and Software: A Festschrift Symposium in Honor of Kokichi Futatsugi*. Springer, May 2014.
15. D. Bjørner. *Domain Engineering: Technology Management, Research and Engineering*. A JAIST Press Research Monograph # 4, 536 pages, March 2009.
16. D. Bjørner. The Role of Domain Engineering in Software Development. Why Current Requirements Engineering Seems Flawed! In *Perspectives of Systems Informatics*, volume 5947 of *Lecture Notes in Computer Science*, pages 2–34, Heidelberg, Wednesday, January 27, 2010. Springer.
17. D. Bjørner and A. Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In *Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59, Heidelberg, July 2010. Springer.
18. D. Bjørner and C. B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer, 1978.
19. D. Bjørner and C. B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

[29] http://www.imm.dtu.dk/~dibj/da-mod-p.pdf
[30] http://www.imm.dtu.dk/~dibj/da-mod-s.pdf

20. R. Casati and A. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.
21. P. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. Database Syst*, 1(1):9–36, 1976.
22. E. A. Feigenbaum and P. McCorduck. *The fifth generation*. Addison-Wesley, Reading, MA, USA, 1st ed. edition, 1983.
23. J. Fitzgerald and P. G. Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.
24. M. Fowler. *Domain Specific Languages*. Signature Series. Addison Wesley, October 20120.
25. C. Fox. *The Ontology of Language: Properties, Individuals and Discourse*. CSLI Publications, Center for the Study of Language and Information, Stanford University, California, ISA, 2000.
26. B. Ganter and R. Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer-Verlag, January 1999. ISBN: 3540627715, 300 pages, Amazon price: US $ 44.95.
27. C. W. George, P. Haff, K. Havelund, A. E. Haxthausen, R. Milne, C. B. Nielsen, S. Prehn, and K. R. Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.
28. C. W. George, A. E. Haxthausen, S. Hughes, R. Milne, S. Prehn, and J. S. Pedersen. *The RAISE Development Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.
29. C. Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985. Published electronically: http://www.usingcsp.com/cspbook.pdf (2004).
30. D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.
31. M. A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England, 1995.
32. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Foda: Feature-oriented domain analysis. Feasibility Study CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, November 1990. http://www.sei.cmu.edu/library/abstracts/reports/90tr021.cfm.
33. E. Luschei. *The Logical Systems of Leśniewksi*. North Holland, Amsterdam, The Netherlands, 1962.
34. N. Medvidovic and E. Colbert. Domain-Specific Software Architectures (DSSA). Power Point Presentation, found on The Internet, Absolute Software Corp., Inc.: `Abs[S/W]`, 5 March 2004.
35. D. H. Mellor and A. Oliver, editors. *Properties*. Oxford Readings in Philosophy. Oxford Univ Press, May 1997. ISBN: 0198751761, 320 pages.
36. R. Prieto-Díaz and G. Arrango. *Domain Analysis and Software Systems Modelling*. IEEE Computer Society Press, 1991.
37. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1998.
38. B. Russel. *"Preface," Our Knowledge of the External World*. G. Allen & Unwin, Ltd., London, 1952.
39. B. Smith. Ontology and the Logistic Analysis of Reality. In G. Haefliger and P. M. Simons, editors, *Analytic Phenomenology*. Dordrecht/Boston/London: Kluwer, Padua, Italy, 1993.

40. J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.