Domain Engineering
A Basis for Safety Critical Software

Dines Bjørner
Fredsvæj 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/˜dibj

Abstract

Before software can be designed we must have a reasonable grasp of the requirements that the software is supposed to fulfil. And before requirements can be prescribed we must have a reasonable grasp of the “underlying” application domain. Domain engineering now becomes a software engineering development phase in which a precise description, desirably formal, of the domain within which the target software is to be embedded. Requirements engineering then becomes a phase of software engineering in which one systematically derives requirements prescriptions from the domain description. (Software design is then the software engineering phase which (also) results in code.) We illustrate the first element, $D$, of this triptych $(D, R, S)$ by an example, Sect. 2, in which we show a description of a pipeline domain where, for example, the operations of pumps and valves are safety critical. We then, Sects. 3–5, summarise the methodological stages and steps of domain engineering. We finally weave considerations of system safety criticality into a section (Sect. 5) on domain facets. We believe this aspect of safety criticality is new: We here connect safety criticality to domain engineering. The study presented here need be deepened. Similar connections need be made to requirements engineering such as it can be “derived” from domain engineering (Bjørner 2008), and to the related software design. That is, three distinct “layers” of safety engineering.

1 Introduction

1.1 A Software Development Triptych

Before software can be designed we must have a reasonable grasp of the requirements that the software is supposed to fulfil. And before requirements can be prescribed we must have a reasonable grasp of the “underlying” application domain. Domain engineering now becomes a software engineering development phase in which a precise description, desirably formal, of the domain within which the target software is to be embedded. Requirements engineering then becomes a phase of software engineering in which one systematically derives requirements prescriptions from the domain description — carving out and extending, as it were, a subset of those domain properties that are computable and for which computing support is required. Software design is then the software engineering phase which results in code (and further documentation). 1

1.2 Domain Description

To us a domain description is a set of pairs of narrative, that is, informal, and formal description texts. The narrative texts should go hand-in-hand with the formal texts; that is, the narrative should be “a reading” of the formalisation; and the formalisation should be an abstraction that emphasises properties of the domain, not some intricate, for example, “executable” model of the domain. These “pairings” will be amply illustrated in Sect. 2. The meaning of a domain description is basically a heterogeneous algebra1, that is: sets of typed entities and a set of typed operations over these. The formalisation language is here the RAISE (George et al. 1995) Specification Language (George et al. 1992); but it could be any of, for example, Alloy (Jackson 2006), Event B (Abrial 2009a), VDM-SL (Bjørner & Jones 1978, 1982, Fitzgerald & Larsen 1998) or Z (Woodcock & Davies 1996). That is, the main structure of the description of the domain endurants, such as we shall advocate it, may, by some readers, be thought of as an ontology (Benjamin & Fensel 1998, Fox 2000). But our concept a domain description is a much wider concept of ontology than covered by (Benjamin & Fensel 1998); it is more in line with (Mellor & Oliver 1997, Fox 2000).

1.3 A Domain Description "Ontology"

We shall, in Sect. 2, give a fairly large example, approximately 3.5 Pages, of a postulated domain of (say, oil or gas) pipelines; the focus will be on endurants: the observable entities that endure, their mereology, that is, how they relate, and their attributes. Perdurants: actions, events and behaviours will be very briefly mentioned.

We shall then, in Sect. 3 on the background of this substantial example, outline the basic principles, techniques and tools for describing domains — focusing only on endurants.

The mathematical structure that is built up when describing a domain hinges on the following elements: there are entities; entities are either endurants or perdurants; endurants are either discrete or continuous; discrete endurants are also called parts; continuous endurants are also called materials; parts are either atomic or composite; parts have unique identifiers, mereologies and attributes; materials have attributes; so entities are what we see and unique identifiers, 1This is just one of the ways in which a domain description differs from an ontology.
mereologies and attributes are entity qualities. A domain description is then composed from one or more part and material descriptions; descriptions of unique part identifiers, part mereologies and part attributes. This structure that, to some, may remind them of an “upper ontology.” Different domain descriptions all basically have the same “upper ontology.”

1.4 Safety Criticality

In Sect. 4 we shall review notions of safety criticality: safety, failure, error, fault, hazard and risk. Other notions will also be briefly characterised: component and system safety, and stake-holder, machine and requirements.

And, finally, in Sect. 5, we shall detail the notion of domain facets. The various domain facets somehow reflect domain views — of logical or algebraic nature — views that are shared across stake-holder groups, but are otherwise clearly separable. It is in connection with the summary explanation of respective domain facets that we identify respective faults and hazards. The presentation is brief. We refer to (Bjørner 2010a) for a more thorough coverage of the notion of domain facets.

1.5 Contribution

We consider the following ideas new: the idea of describing domains before prescribing requirements (but see (Bjørner 2006c, Part IV, 2006), (Bjørner 2007, 2007), (Bjørner 2010a, written in 2007, published in 2010), (Bjørner 2008, 2008), (Bjørner 2010b, 2011a, 2010), and (Bjørner 2014b, 2014)), and the idea of enumerating faults and hazards as related to individual facets. For the latter “discovery” we thank the organisers of ASSC 2014, notably Prof. Clive Victor Boughton.

2 An Example

Our example is an abstraction of pipeline system endurants. The presentation of the example reflects a rigorous use of the domain analysis & description method outlined in Sect. 3, but is relaxed with respect to not showing all — one could say intermediate — analysis steps and description texts, but following stoichiometry ideas from chemistry makes a few short-cuts here and there. The use of the “stoichiometrical” reductions, usually skipping intermediate endurant sorts, ought properly be justified in each step — and such is advised in proper, industry-scale analyses & descriptions.

To guide your intuition with respect to what a pipeline system might be we suggest some diagrams and some pictures. See Figs. 1 and 2.

Figure 1: Pipelines. Flow is right-to-left in left figure, but left-to-right in right figure.

The description only covers a few aspects of endurants.

Figure 2: Pump, pipe and valve pipeline units

2.1 Parts

1. A pipeline system contains a set of pipeline units and a pipeline system monitor.

2. The well-formedness of a pipeline system depends on its mereology (cf. Sect. 2.2.3) and the routing of its pipes (cf. Sect. 2.3.2).

3. A pipeline unit is either a well, a pipe, a pump, a valve, a fork, a join, or a sink unit.

4. We consider all these units to be distinguishable, i.e., the set of wells, the set pipe, etc., the set of sinks, to be disjoint.

5. Each pipeline unit is uniquely distinguished by its unique unit identifier.

6. We is the subtype (i.e., subset) of well-formed PLS, i.e., well-formedness defines only well-formed units.

2.2 Part Identification and Mereology

2.2.1 Unique Identification.

5. Each pipeline unit is uniquely distinguished by its unique unit identifier.

2.2.2 Unique Identification.

5. Each pipeline unit is uniquely distinguished by its unique unit identifier.

6. We is the subtype (i.e., subset) of well-formed PLS, i.e., well-formedness defines only well-formed units.

2.2.3 Unique Identification.

5. Each pipeline unit is uniquely distinguished by its unique unit identifier.

6. We is the subtype (i.e., subset) of well-formed PLS, i.e., well-formedness defines only well-formed units.
2.2.2 Unique Identifiers.

6. From a pipeline system one can observe the set of all unique unit identifiers.

value
7. \( \forall \text{pls} : \text{PLS}, \exists \text{UID}(\text{pls}) \)

5. \( \{ u, u' \} \subseteq \text{obs}_\text{Us}(\text{pls}) \) \( \Rightarrow \) \( u \neq u' \) \( \Rightarrow \) \( \text{uid}_\text{UID}(u) \neq \text{uid}_\text{UID}(u') \)

2.2.3 Mereology

8. Each unit is connected to zero, one or two other existing (formula line 8x.) input units and zero, one or two other existing (formula line 8x.) output units as follows:

- A well unit is connected to exactly one output unit (and, hence, has no “output”).
- A pipe unit is connected to exactly one input unit and one output unit.
- A pump unit is connected to exactly one input unit and one output unit.
- A valve is connected to exactly one input unit and two distinct output units.
- A fork is connected to exactly two distinct input units and one output unit.
- A sink is connected to exactly one input unit (and, hence, has no “output”).

8. \( \text{UID}_\text{PLS} : \text{PLS} \rightarrow \text{UID-set} \)

6. \( \text{UID}_\text{PLS}(\text{pls}) \equiv \{ \text{uid}_\text{UID}(u) \mid u : U \in \text{obs}_\text{Us}(\text{pls}) \} \)

7. We can prove that the number of unique unit identifiers of a pipeline system equals that of the units of that system.

theorem:
7. \( \forall \text{pls} : \text{PLS} \rightarrow \text{card obs}_\text{Us}(\text{pl}) = \text{card xtr}_\text{Us}(\text{pls}) \)

2.3 Part Concepts

An aspect of domain analysis & description that was not covered in Sects. 2.1–2.2 was that of derived concepts. Example pipeline concepts are routes, acyclic or cyclic, circular, etcetera. In expressing well-formedness of pipeline systems one often has to develop subsidiary concepts such as these by means of which well-formedness is then expressed.

2.3.1 Pipe Routes.

9. A route (of a pipeline system) is a sequence of connected units (of the pipeline system).

10. A route descriptor is a sequence of unit identifiers and the connected units of a route (of a pipeline system).

value
9. \( R' = U^{\neq 21} \)

10. \( R = \{ r : \text{Route} \mid \text{WF Route}(r) \} \)

10. \( \text{RD} = U^{\text{RD}} \)

10. \( \forall \text{rd} : \text{RD} \rightarrow \exists r : \text{rd} = \text{descriptor}(r) \)

10. \( \text{descriptor}(r) \equiv \{ \text{uid}_\text{UID}(r[i]) \mid i : \text{Nat} \cdot 1 \leq i \leq \text{len } r \}

11. Two units are adjacent if the output unit identifiers of one shares a unique unit with the input identifiers of the other.

value
11. \( \text{adjacent} : U \times U \rightarrow \text{Bool} \)

11. \( \text{adjacent}(u,u') \equiv \)

11. \( \text{let } \) \( \{ \text{ouis}=\text{mereco}_U(u),\{\text{iuis}=\text{mereco}_U(u')\} \) \( \text{in} \)

11. \( \text{ouis} \cap \text{iuis} \neq \{ \} \) \( \text{end} \)

12. Given a pipeline system, \( \text{pls} \), one can identify the (possibly infinite) set of (possibly infinite) routes of that pipeline system.

a. The empty sequence, \( \langle \rangle \), is a route of \( \text{pls} \).

b. Let \( u, u' \) be any units of \( \text{pls} \), such that an output unit identifier of \( u \) is the same as an input unit identifier of \( u' \) then \( \langle u, u' \rangle \) is a route of \( \text{pls} \).

c. If \( r \) and \( r' \) are routes of \( \text{pls} \) such that the last element of \( r \) is the same as the first element of \( r' \), then \( \hat{r}^{-\text{th}} r' \) is a route of \( \text{pls} \).

d. No sequence of units is a route unless it follows from a finite (or an infinite) number of applications of the basis and induction clauses of Items 12a–12e.

\[ \text{uid}_\text{UID} \] is the unique identifier observer function for parts \( u \).

It is total. \( \text{uid}_\text{UID}(u) \) yields the unique identifier of \( u \).

\[ \text{UID}_\text{PLS} \] is a total function. It extracts all unique unit identifiers of a pipeline system.

\[ \text{UID}_\text{PLS} \] is the unique identifier observer function for parts \( u \).

It is total. \( \text{uid}_\text{UID}(u) \) yields the unique identifier of \( u \).

\[ \text{UID}_\text{PLS} \] is a total function. It extracts all unique unit identifiers of a pipeline system.
value
12. Routes: PLS → RD-infset
12a. let rs = (∅ ∪ 
12b. \{uid/UI(u),uid/UI(u')|u,u':U•{u,u'}
12c. \⊆ obsUs(pls) ∧ adjacent(u,u')\}
12d. \∪ \{r''\r'\r:R•\{r,r'\}⊆rs\})

2.3.2 Well-formed Routes.
13. A route is acyclic if no two route positions reveal
the same unique unit identifier.
value
13a. acyclicRoute: R → Bool
13b. acyclicRoute(r) ≡ 
13c. \sim \exists i,j:Nat\{i,j]\subseteq\text{inds} r \land i\neq j \land r[i]=r[j]

14. A pipeline system is well-formed if none of its
routes are circular (and all of its routes embedded
in well-to-sink routes).
value
14a. wfroutes: PLS → Bool
14b. wfroutes(pls) ≡ 
14c. non_circular(pls) ∧ 
14d. embedded_in_well_to_sink_Routes(pls)

15. We define well-formedness in terms of well-to-
sink routes, i.e., routes which start with a well
unit and end with a sink unit.
value
15a. well_to_sink_Routes: PLS → R-set
15b. well_to_sink_Routes(pls) ≡ 
15c. let rs = Routes(pls) in 
15d. \{r|r:R•r \in routes(p)/acyclicRoute(r)

16. A pipeline system is well-formed if all of its routes
are embedded in well-to-sink routes.
value
16a. embedded_in_well_to_sink_Routes: PLS → Bool
16b. embedded_in_well_to_sink_Routes(pls) ≡ 
16c. let wsrs = well_to_sink_Routes(pls) in 
16d. \forall r:R•r \in Routes(pls) ⇒ 
16e. \exists r\r:R,i,j:Nat• 
16f. \r' \in wsrs 
16g. \land \{i,j\}⊆\text{inds} r'\land i\leq j 
16h. \land r = \{r'[k]|k:Nat•i\leq k\leq j\}

2.3.3 Embedded Routes.
17. For every route we can define the set of all its
embedded routes.
value
17a. embedded_Routes: R → R-set
17b. embedded_Routes(r) ≡ 
17c. \{(r'[k]|k:Nat•i\leq k\leq j\}
17d. \mid i,j:Nat•i\{i,j\}⊆\text{inds}(r) \land i\leq j\)

2.3.4 A Theorem.
18. The following theorem is conjectured:

a the set of all routes (of the pipeline system)
b is the set of all well-to-sink routes (of a
pipeline system) and
c all their embedded routes

18a. \forall pls:PLS •
18b. let rs = Routes(pls),
18c. wsrs = well_to_sink_Routes(pls) in 
18d. rs = 
18e. \subseteq wsrs \cup 
18f. \{\{r'[r':R•r' \in \text{embedded_Routes}(r')\}
18g. \mid r''\r':R \subseteq wsrs\} 
18h. end

2.4 Materials
19. The only material of concern to pipelines is the
gas or the liquid which the pipes transport.

2.5 Attributes
2.5.1 Part Attributes.
20. These are some attribute types:

a estimated current well capacity (barrels of
oil, etc.),
b pipe length,
c current pump height,
d current valve open/close status and
f flow (e.g., volume/second).

20a. WellCap
20b. LEN
20c. Height
20d. ValSta • open | close
20e. Flow

21. Flows can be added (also distributively) and
subtracted, and
22. flows can be compared.
value
21a. ⊕, ⊖: Flow × Flow → Flow
21b. ⊕: Flow × Flow → Flow
22. \langle,\leq,\neq,\geq\rangle: Flow × Flow → Bool

23. Properties of pipeline units include

a estimated current well capacity (barrels of
oil, etc.),

20Gaseous materials include: air, gas, etc.
21Liquid materials include water, oil, etc.
22The description of this paper is relevant only to gas or oil pipelines.
b) pipe length,
c) current pump height,
d) current valve open/close status,
e) current laminar in-flow at unit input,
f) current laminar in-flow leak at unit input,
g) maximum laminar guaranteed in-flow leak at unit input,
h) current laminar leak unit interior,
i) current laminar flow in unit interior,
j) maximum laminar guaranteed flow in unit interior,
k) current laminar out-flow at unit output,
l) current laminar out-flow leak at unit output,
m) maximum guaranteed laminar out-flow leak at unit output.

value
23a. attrW ellCap: We \rightarrow WellCap
23b. attrLEN: Fl \rightarrow LEN
23c. attrHeight: Pu \rightarrow Height
23d. attrValSta: V a \rightarrow VaSta
23e. attrIn_Flow: U \rightarrow UI \rightarrow Flow
23f. attrIn_Leak: U \rightarrow UI \rightarrow Flow
23g. attrMax-In_Leak: U \rightarrow UI \rightarrow Flow
23h. attrBody_In_Flow: U \rightarrow Flow
23i. attrBody_In_Leak: U \rightarrow Flow
23j. attrMax_In_Flow: U \rightarrow Flow
23k. attrOut_Flow: U \rightarrow UI \rightarrow Flow
23l. attrOut_In_Leak: U \rightarrow UI \rightarrow Flow
23m. attrMax_Out_In_Leak: U \rightarrow UI \rightarrow Flow

2.5.2 Flow Laws.
24. “What flows in, flows out!” For laminar flows: for any non-nil and non-sink unit the sums of input leaks and in-flows equals the sums of unit and output leaks and out-flows.

Law:
24. \forall u[U \cup V] \cdot \sum_{in} \text{leaks}(u) \oplus \sum_{in} \text{flows}(u) = \sum_{body} \text{Max-In_Leak}(u) \oplus \sum_{out} \text{out_leaks}(u) \oplus \sum_{out} \text{flows}(u)

value
sum_{in} \text{leaks}: U \rightarrow Flow
sum_{in} \text{leaks}(u) \equiv
let \{(\text{uis}) = \text{mero}_U(u) \in \text{in} \mid \text{uis} \in \text{uis}\} end
\sum_{in} \text{flows}: U \rightarrow Flow
\sum_{in} \text{flows}(u) \equiv
let \{(\text{uis}) = \text{mero}_U(u) \in \text{in} \mid \text{uis} \in \text{uis}\} end
\sum_{out} \text{out_leaks}: U \rightarrow Flow
\sum_{out} \text{out_leaks}(u) \equiv
let \{(\text{ois}) = \text{mero}_U(u) \in \text{out} \mid \text{ois} \in \text{ois}\} end
\sum_{out} \text{flows}: U \rightarrow Flow
\sum_{out} \text{flows}(u) \equiv
let \{(\text{ois}) = \text{mero}_U(u) \in \text{out} \mid \text{ois} \in \text{ois}\} end

25. “What flows out, flows in!” For laminar flows: for any adjacent pairs of units the output flow at one unit connection equals the sum of adjacent unit leak and in-flow at that connection.

Law:
25. \forall u,u'[U \cap \text{adjacent}(u,u')] \Rightarrow
25. \text{let } \{(\text{ois}) = \text{mero}_U(u) \in \text{in} \mid \text{ois} \in \text{ois}\} in
25. \text{let } \{(\text{uis}) = \text{mero}_U(u) \in \text{in} \mid \text{uis} \in \text{uis}\} end
25. \text{uid}_U(u') \in \text{ois} \land \text{uid}_U(u) \in \text{uis} \land
25. \text{attr_out_Flow}_\text{L}(u)(\text{uid}_U(u')) =
25. \text{attr_in_Leak}_\text{L}(u)(\text{uid}_U(u)) =
25. \text{\oplus att}_\text{attr_out_Flow}_\text{L}(u)(\text{uid}_U(u)) \text{ end}

2.5.3 Open Routes.
26. A route, r, is open
   a) if all valves, v, of the route are open and
   b) if all pumps, p, of the route are pumping.

value
26. is_open: R \rightarrow Bool
26. is_open(r) \equiv
26a. \forall \text{mkPu}(p):Pu \cdot \text{mkPu}(p) \in \text{elems} r
26b. \Rightarrow is\text{pumping}(p) \land
26b. \forall \text{mkVa}(v):Va \cdot \text{mkVa}(v) \in \text{elems} r
26b. \Rightarrow is\text{open}(v)

2.6 Domain Perdurants

2.6.1 Actions.
We shall not formalise any specific actions. Informal examples of actions are: opening and closing a well, start and stop pumping, open and close valves, opening and closing a sink and sense current unit flow.

2.6.2 Events.
We shall not formalise any specific events. Informal examples of events are: empty well, full sink, start pumping signal to pump with no liquid material, pump ignores start/stop pumping signal, valve ignores opening/closing signal, excessive to catastrophic unit leak, and unit fire or explosion.

2.6.3 Behaviours.
We shall not formalise any specific behaviours. Informal examples of behaviours are: start pumping and opening up valves across a pipeline system, and stop pumping and closing down valves across a pipeline system.

3 Basic Domain Description
In this section and in Sect.5 we shall survey basic principles of describing, respectively, domain intrinsics and other domain facets.

By an entity we shall understand a phenomenon that can be observed, i.e., be seen or touched by humans, or that can be conceived as an abstraction of an entity.

Example: Pipeline systems, units and materials are entities (Page2, Item1).

The method can thus be said to provide the domain analysis prompt: is_entity where is_entity(θ) holds if θ is an entity.

A domain is characterised by its observable, i.e., manifest entities and their 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.
Example: Unique identifiers (Page 2, Item 5.), mereology (Page 3, Item 8.) and the well capacity (Page 4, Item 20a.), pipe length (Page 4, Item 20b.), current pump height (Page 4, Item 20c.), current valve open/close status (Page 4, Item 20d.) and flow (Page 4, Item 20e.) attributes are qualities.

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.

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 in time. Thus the method provides a domain analysis prompt: is_endurant where is_endurant(e) holds if entity e is an endurant.

By a perdurant entity (or just, a 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: is_perdurant where is_perdurant(e) holds if entity e is a perdurant.

By a discrete endurant we shall understand something which is separate or distinct in form or concept, consisting of distinct or separate parts.

By a continuous endurant we shall understand something which is prolonged without interruption, in an unbroken series or pattern.

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 a continuous endurant.

3.1 Endurant Entities

We distinguish between endurant and perdurant entities.

Parts and Materials: The manifest entities, i.e., the endurants, are called parts, respectively materials. We use the term part for discrete endurants, that is:

- We use the term material for continuous endurants.
- Discrete endurants are either atomic or composite.

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 atomic part.

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.

Example: Pipeline units, PLS, and the set, Us, of pipeline units are considered composite entities.

3.1.1 Part Observers.

From atomic parts we cannot observe any sub-parts. But from composite parts we can. For composite parts, p, the domain description prompt observe-part sorts(p) yields some formal description text according to the following schema:

<table>
<thead>
<tr>
<th>type</th>
<th>P1, P2, ..., Pn</th>
<th>value</th>
<th>obs_P1: P = P1, obs_P2: P = P2, ..., obs_Pn: P = Pn</th>
</tr>
</thead>
</table>

where sort names P1, P2, ..., Pn 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 of sorts.

Example: Three formula lines (Page 2, Items 1.) illustrate the basic sorts (CLS, US, U, M) and observers (obs_US, obs_M) of pipeline systems.

3.1.2 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: 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:

* either

<table>
<thead>
<tr>
<th>type</th>
<th>P1, P2, ..., PN, T = E(P1,P2,...,PN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>obs_T: P → T</td>
</tr>
</tbody>
</table>

  where 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.

* or:

<table>
<thead>
<tr>
<th>type</th>
<th>T = P1</th>
<th>P2</th>
<th>...</th>
<th>PN</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>obs_T: P → T</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: obs_T: P → T is exemplified by obs_US: PS → U-set (Page 2, Item 1.), T = P1 | P2 | ... | PN by We | Pu | Va | Fo | Jo | Si (Page 2, Item 3.) and P1 :: mkP1(P1), P2 :: mkP2(P2), ..., PN :: mkPN(Pn) by (Page 2, Item 4.)

---

29 This REL type clause defines P1, P2, ..., Pn, to be sorts.
30 This REL value clause defines n function values. All from type P into some type Pn.
31 The concrete type definition T = μ(E(P1,P2,...,PN)) defines type T to be the set of elements of the type expressed by type expression μ(E(P1,P2,...,PN)).
32 obs_T is a function from any element of P to some element of T.
33 The domain analysis prompt: sorts_of(t) yields a subset of {P1,P2,...,PN}.
34 A | B is the union type of types A and B.
35 Type definition A :: mk(A) defines type A to be the set of elements mk(A) where b is any element of type B.
36 obs_T is a function from any element of P to some element of T.
3.1.3 Material Observers.

Some parts \( p \) of sort \( P \) may contain material. The domain analysis prompt has\_material\( (p) \) 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:

\[
\text{type} \ M_1, M_2, \ldots, M_m; \\
\text{value} \ \text{obs}_{M_1}: P \rightarrow M_1, \ \text{obs}_{M_2}: P \rightarrow M_2, \\
\ldots, \ \text{obs}_{M_m}: P \rightarrow M_m; \\
\]

where values, \( m_i \), of type \( M_i \) satisfy is\_material\( (m) \) for all \( i \); and where \( M_1, M_2, \ldots, M_m \) must be disjoint sorts.

Example: We refer to Sect. 2.4 (Page 4, Item 19.)

3.2 Endurant Qualities

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).

3.2.1 Unique Part Identifiers.

Without loss of generality we can assume that every part has a unique identifier\( ^{37} \). 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:

\[
\text{type} \ \text{PI}; \\
\text{value} \ \text{uid\_P}: P \rightarrow \text{PI}; \\
\]

Example: We refer to Page 2, Item 5.

3.2.2 Part Mereology.

By mereology (Luschei 1962) 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 attributes.

The domain analysis prompt: has\_mereology\( (p) \) holds if the part \( p \) is related to some others parts \( (p_a, p_b, \ldots, p_e) \). The domain description prompt: observe\_mereology\( (p) \) can then be invoked and yields some formal description text according to the following schema:

\[
\text{type} \ \text{MT} = \mathcal{E}(\text{PI}_A, \text{PI}_B, \ldots, \text{PI}_C); \\
\text{value} \ \text{mereo\_P}: P \rightarrow \text{MT}; \\
\]

where \( \mathcal{E}(\ldots) \) 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.

Example: We refer to Items 8.–8g, Pages 3–3

\[ ^{37} \text{That is, has\_unique\_identifier\( (p) \) for all parts \( p \).} \]

3.2.3 Part and Material Attributes.

Attributes are what really endows parts with qualities. The external properties\( ^{38} \) 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.

By an endurant 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 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 observe\_attributes\( (p) \) yields some formal description text according to the following schema:

\[
\text{type} \ A_1, A_2, \ldots, A_n; \\
\text{value} \ \text{attr}_{A_1}: P \rightarrow A_1, \\
\text{attr}_{A_2}: P \rightarrow A_2, \\
\ldots, \\
\text{attr}_{A_n}: P \rightarrow A_n; \\
\]

Example: We refer to Sect. 2.5 Pages 4–5

3.3 Perdurant Entities

We shall cover the principles, tools and techniques for “discovering”, analysing and describing domain actions, events and behaviours to anywhere the detail with which the “corresponding” principles, tools and techniques were covered for endurants. But we shall summarise one essence for the description of perdurants.

There is a notion of state. Any composition of parts having dynamic qualities can form a state. Dynamic qualities are qualities that may change. Examples of such qualities are the mereology of a part, and part attributes whose value may change.

There is the notion of function signature. A function signature, \( f: A (\rightarrow \rightarrow) R \), gives a name, say \( f \), to a function, expresses a type, say \( T_A \), of the arguments of the function, expresses whether the function is total (\( \rightarrow \) or partial (\( \rightarrow \)), and expresses a type, say \( T_R \), of the result of the function.

There is the notion of channels of synchronisation & communication between behaviours. Channels have names, e.g., \( ch, ch_1, ch_2 \). Channel names appear in the signature of behaviour functions: value \( b: A \rightarrow \text{in} \ ch \text{out} \ ch_o \ R \). \( \text{in} \ ch \text{out} \) indicates that behaviour \( b \) may express willingness to communicate an input message over channel \( ch \); and \( \text{out} \ ch_o \) indicates that behaviour \( b \) may express an offer to communicate an output message over channel \( ch_o \).

There is a notion of function pre/post-conditions.

A function pre-condition is a predicate over argument values. A function post-condition is a predicate over argument and result values.

Action signatures include states, \( \Sigma \), in both arguments, \( A \times \Sigma \), and results, \( \Sigma : f: A \times \Sigma \rightarrow \Sigma \); \( f \) denotes a function in the function space \( A \times \Sigma \rightarrow \Sigma \). Action pre/post-conditions:

\[
\text{value} \\
\text{f(a,} \sigma) \text{ as } \sigma'; \\
\text{pre: } P_1(a,} \sigma); \\
\text{post: } Q_1(a,} \sigma, \sigma') \\
\]

\[ ^{38} \text{is\_discrete, is\_continuous, is\_atomic, is\_composite, has\_material.} \]
have predicates $P_\pi$ and $Q_\pi$ delimit the value of $\pi$ within that function space.

Event signatures are typically predicates from pairs of before and after states: $e: \Sigma \times \Sigma \rightarrow \text{Bool}$. Event pre/post-conditions

\[
\text{value} \quad e: \Sigma \times \Sigma \rightarrow \text{Bool}; \\
\text{e}(\sigma, \sigma') \equiv P_\pi(\sigma) \land Q_\pi(\sigma')
\]

have predicates $P_\pi$ and $Q_\pi$ delimit the value of $e$ within the $\Sigma \times \Sigma \rightarrow \text{Bool}$ function space; $P_\pi$ characterises states leading to event $e$: $Q_\pi$ characterises states, $\sigma'$, resulting from the event caused by $\sigma$.

In principle we can associate a behaviour with every part of a domain. Parts, $p$, are characterised by their unique identifiers, $p: \text{PI}$ and a state, $\text{attrs:ATTRS}$. We shall, with no loss of generality, assume part behaviours to be never-ending. The unique part identifier, $p: \text{PI}$, and its part mereology, say $\{p_1, p_2, ..., p_n\}$, determine a number of channels $\{\text{chs}[p_1, p_2]|\{1, 2, ..., n\}\}$ to communicate messages of type $M$. Behaviour signatures:

\[
b: p: \text{PI} \times \text{ATTR} \rightarrow \text{in}_\text{chs} \text{ out}_\text{chs} \text{ Unit}
\]

then have input channel expressions in $\text{chs}$ and output channel expressions out $\text{chs}$ be suitable predicates over $\{\text{chs}[p_1, p_2]|\{1, 2, ..., n\}\}$. Unit designate that $b$ denote a never-ending process. We omit dealing with behaviour pre-conditions and invariants.

4 Interlude

We have covered one aspect of the modelling of one set of domain entities, the intrinsic facets of endurants. For the modelling of perdurants we refer to (Björner 2010b, 2011a, 2014a). In the next section, Sect. 5, we shall survey the modelling of further domain facets. We shall accompany this survey to a survey of safety issues. To do so in a reasonably coherent way we need establish a few concepts: the safety notions of failure, error and fault: the notion of stake-holder and the notion of requirements.

4.1 Safety-related Concepts

Some characterisations are:

Safety: By safety, in the context of a domain being dependable, we mean some measure of continuous delivery of service of either correct service, or incorrect service after benign failure, that is: measure of time to catastrophic failure.

Failure: A domain failure occurs when the delivered service deviates from fulfilling the domain function, the latter being what the domain is aimed at (Randell 2003).

Error: An error is that part of a domain state which is liable to lead to subsequent failure. An error affecting the service is an indication that a failure occurs or has occurred (Randell 2003).

Fault: The adjudged (i.e., the ‘so-judged’) or hypothesised cause of an error is a fault (Randell 2003).

Hazard: A hazard is any source of potential damage, harm or adverse health effects on something or someone under certain conditions at work.

Risk: A risk is the chance or probability that a person will be harmed or experience an adverse health effect if exposed to a hazard. It may also apply to situations with property or equipment loss.

Faults and Hazards: The concept of hazard is not the same as the concept of fault. “System safety takes a large view of hazards than just failures (Leveson 2003): Hazards are not always caused by failures, and all failures do not cause hazards. Serious accidents have occurred while system components were all functioning exactly as specified, that is, without failure. If failures only are considered in a safety analysis, many potential accidents will be missed. In addition, the engineering approaches to preventing failures (increasing reliability) and preventing hazards (increasing safety) are different and sometimes conflict.”

4.2 System and Component Safety

There appears to be a number of safety concepts (Leveson 2003): component safety, industrial safety, reliability, and system safety. We shall focus on component and system safety.

Component: By a component we shall understand basically the same as an atomic part together with actions, events and behaviours whose state is anchored in one or more attributes of that part, such that these actions, etc., do nor involve other component or sub-system states. That is, “componentry” excludes considerations of shared attributes.

System: By a system or sub-system we shall understand basically the same as a composite part together with actions, events and behaviours whose state is evidenced in one or more attributes of that part as well as of one or more other parts. That is, “system-hood” presumes considerations of shared attributes.

System Safety: “The primary concern of system safety (Leveson 2003) is the management of hazards: their identification, evaluation, elimination, and control through analysis, design and management procedures.” “System safety deals with systems as a whole rather than with subsystems or components (Leveson 2003): Safety is an emergent property of systems, not a component property. One of the principle responsibilities of system safety is to evaluate the interfaces between the system components and determine the effects of component interaction, where the set of components includes humans, machines, and the environment.”

The system interfaces are given by the mereology.

Component Safety: For a component, that is, an atomic part, we can, at most, speak of faults when considering safety.39

4.3 Stake-holder

By a domain stake-holder we shall understand a person, or a group of persons, “united” somehow in their common interest in, or dependency on the domain; or an institution, an enterprise, or a group of such, (again) characterised (and, again, loosely) by their common interest in, or dependency on the domain.40

Examples: The following are examples of pipeline stake-holders: the owners of the pipeline, the oil or gas companies using the pipeline, the pipeline managers and workers, the owners and neighbours of the lands occupied by the pipeline, the citizens possibly

39 The borderline between hazards that are not faults and faults is too vague.
worried about gas- or oil pollution, the state authorities regulating and overseeing pipelining, etcetera. □

4.4 Machines and Requirements

4.4.1 Machine.

By the machine we shall understand the combination of hardware, say computers and communication, and software.

4.4.2 Requirements.

By a requirements we understand (cf. IEEE Standard 610.12 (IEEE Computer Society 1990)): “A condition or capability needed by a user to solve a problem or achieve an objective” □ We shall think only of requirements as requirements to a machine. We can now “repeat” the definitions of safety, failure, error and fault given above, but now with the term ‘domain’ replaced by the term ‘machine’ (sometimes with the term ‘domain+machine’). This then becomes the context in which most safety criticality is discussed.

We shall not cover requirements in this paper. We refer to (Bjørner 2008). That paper describes how to “derive” systematically, but, of course, not automatically major parts of requirements prescriptions from a domain descriptions. Thus we shall not cover the classical approach to safety analysis. Instead we shall cover what we think is a novel approach to safety analysis. One in which first get an as complete as possible overview of “all” safety aspects of a domain.

5 Domain Facets and Safety Criticality

5.1 Introductory Notions

5.1.1 Facet.

By a domain facet we shall understand one amongst a finite set of generic ways of analysing a domain: a view of the domain, such that the different facets cover conceptually different views, and such that these views together cover the domain □

We shall distinguish between the following facets: intrinsics, support technologies, human behaviour, rules & regulations and organisation & management.

In the following we refer to respective subsections of (Bjørner 2010a) should the reader wish further elaborations of the facet concept.

5.1.2 Safety Criticality.

Safety critical systems are those systems whose failure may result in the loss of life, significant property damage or damage to the environment. 41

For each of the domain facet categories we shall look for a corresponding, domain-specific category of hazards. That is, we shall view safety criticality in potentially three steps: from the point of view of the domain in which a computing system is to be inserted, hence first developed, from the point of view of the requirements prescribed for such a system, and from the point of view of the machine (i.e., hardware + software) design of that system. In this paper we shall only consider the first step.

5.2 Intrinsics

By domain intrinsics (Bjørner 2010a, 1.4.1, 11–15) 42 we shall understand those phenomena and concepts of a domain which are basic to any of the other facets (listed earlier and treated, in some detail, below), with such domain intrinsics initially covering at least one specific, hence named, stake-holder view □

Example: The example of Sect.2 focused on the intrinsics of pipeline systems as well as some derived concepts (routes etc.). □

Hazards: The following are examples of hazards based solely on the intrinsics of the domain: environmental hazards: destruction of one or more pipeline units due to an earthquake, an explosion, a fire or something similar; see an example of Sect. 2. More specifically to the facet of support technology, as a concept, is related to actions of specific parts; that is, a part may give rise to one or more support technologies, and we say that the support technologies ‘reside’ in those parts.

Examples: wells are, in the intrinsics facet description abstracted as atomic units but in real instances they are complicated (composite) entities of pumps, valves and pipes; pumps are similarly, but perhaps not as complicated complex units; valves likewise; and sinks are, in a sense, the inverse of wells □

Faults: a pump may fail to respond to a stop pump signal; and a valve may fail to respond to an open valve signal □ I think it is fair to say that most papers on the design of safety critical software are on software for the monitoring & control of support technology.

Describing causes of errors is not simple. With today’s formal methods tools and techniques 44 quite a lot can be formalised — but not all!!

□42(Bjørner 2010a, 1.4.1, 11–15) refers to publication (Bjørner 2010a), Sect. 1.4.1, Pages 11–15.

□43We refer to the example of Sect. 2. More specifically to the well-formedness of pipeline systems as expressed in wPLS (Page 2, Item 2). We express hazards of the intrinsics of pipeline systems by named predicates over PLS’ and not PLS.

□44These tools and techniques typically include two or more formal specification languages, for example: VDM (Bjørner & Jones 1978, 1982, Fitzgerald & Larsen 1998), DC (Zhou & Hansen 2004), Event-B (Abrial 2006), RAISE/RLS (George et al. 1995, 1992, Bjørner 2000a,b,c), TLA+ (Lamport 2002) and Alloy (Jackson 2006); one or more theorem proving tools, for example: ACL (Kaufmann et al. 2000b,a), Coq (Huet & Castéran 2004), Isabelle/HOL (Nipkow et al. 2002), StéP (Bjørner et al. 2000), PVS (Shankar et al. 1999) and Z (Bjørner et al. 2014); a model-checker, for example: SMV (Clarke et al. January 2000) and SPIN/Promela (Holmann 2003); and other such tools and techniques; cf. (Bjørner & Havelund 2014).
5.4 Human Behaviour

A proper domain description includes humans as both (usually atomic) parts and the behaviours that we (generally) “attach” to parts.

Examples: The human operators that operate wells, valves, pumps and sinks; check on pipeline units; decide on the flow of material in pipes, etcetera.

By domain human behaviour (Bjørner 2010a, 1.4.6, 27–29) we shall understand any of a quality spectrum of humans45 carrying out assigned work: from (i) careful, diligent and accurate, via (ii) sloppy dispatch, and (iii) delinquent work, to (iv) outright criminal pursuit.

Typically human behaviour focus on actions and behaviours that are carried out by humans. The intrinsics description of actions and behaviours focus solely on intended, careful, diligent and accurate performance.

Hazards: This leaves “all other behaviours” as hazards! Proper hazard analysis, however, usually explicitly identifies failed human behaviours, for example, as identified deviations from described actions etc. Hazard descriptions thus follow from “their corresponding” intrinsics descriptions.

5.5 Rules & Regulations

Rules and regulations (Bjørner 2010a, 1.4.4, 24–26) come in pairs (R_u, R_c).

5.5.1 Rules.

By a domain rule we shall understand some text which prescribes how people are, or equipment is, “expected” (for “…” see below) to behave when dispatching their duty, respectively when performing their function.

Example: There are rules for operating pumps. One is: A pump, p, on some well-to-sink route r = \( r'' (p) r'' \), may not be started if there does not exist an open, embedded route \( r'' \) such that \( (p) r'' \) ends in an open sink.

Hazards: when stipulating “expected”, as above, the rules more or less implicitly express also the safety criticality: that is, when people are, or equipment is, behaving erroneously.

Example: A domain rule which states, for example, that a pump, p, on some well-to-sink route \( r = r'' (p) r'' \), may be started even if there does not exist an open, embedded route \( r'' \) such that \( (p) r'' \) ends in an open sink is a hazardous rule.

Modelling Rules: We can model a rule by giving it both a syntax and a semantics. And we can choose to model the semantics of a rule, \( \mathcal{R}_u \), as a predicate, \( \mathcal{P} \), over pairs of states: \( \mathcal{P} : \Sigma \times \Sigma \rightarrow \mathbb{Bool} \). That is, the meaning, \( \mathcal{M} \), of \( \mathcal{R}_u \) is \( \mathcal{P} \). An action or an event has changed a state \( \sigma \) into a state \( \sigma' \). If \( \mathcal{P}(\sigma, \sigma') \) is true it shall mean that the rule has been obeyed. If it is false it means that the rule has been violated.

5.5.2 Regulations.

By a domain regulation we shall understand some text which “prescribe” (“…” see below) the remedial actions that are to be taken when it is decided that a rule has not been followed according to its intention.

Example: There are regulations for operating pumps and valves: Once it has been discovered that a rule is hazardous there should be a regulation which (i) starts an administrative procedure which ensures that the rule is replaced; and (ii) starts a series of actions which somehow brings the state of the pipeline into one which poses no danger and then applies a non-hazard rule.

Hazards: when stipulating “prescribe”, regulations express requirements to emerging hardware and software.

Modelling Regulations: We can model a regulation by giving it both a syntax and a semantics. And we can choose to model the semantics of a regulation, \( \mathcal{R}_c \), as a state-transformer, \( \mathcal{S} \), over pairs of states: \( \mathcal{S} : \Sigma \times \Sigma \rightarrow \Sigma \). That is, the meaning, \( \mathcal{M} \), of \( \mathcal{R}_c \) is \( \mathcal{S} \). A state-transformation \( \mathcal{S}(\sigma, \sigma') \) for rule \( \mathcal{R}_c \) results in a state \( \sigma'' \) where: if \( \mathcal{P}(\sigma, \sigma') \) is true then \( \sigma' = \sigma'' \), else \( \sigma'' \) is a corrected state such that \( \mathcal{P}(\sigma, \sigma'') \) is true.

5.5.3 Discussion.

Where do rules & regulations reside?" That is, “Who checks that rules are obeyed?” and “Who ensures that regulations are applied when rules fail?” Are some of these checks and follow-ups relegated to humans (i.e., parts) or to machines (i.e., “other” parts)? that is, to the behaviour of part processes? The next section will basically answer those questions.

5.6 Organisation & Management

To (Bjørner 2010a, 1.4.3, 17–21) properly appreciate this section we need remind the reader of concepts introduced earlier in this paper. With parts we associate mereologies, attributes and behaviours. Support technology is related to actions and these again focused on parts. Humans are often modelled first as parts, then as their associated behaviour. It is out of this seeming jigsaw puzzle of parts, mereologies, attributes, humans, rules and regulations that we shall now form and model the concepts of organisation and management.

5.6.1 Organisation.

By domain organisation we shall understand one or more partitionings of resources where resources are usually representable as parts and materials and where usually a resource belongs to exactly one partition; such that n such partitionings typically reflects strategic46 (say partition \( \pi_s \)), tactical47 (say partition \( \pi_t \)), respectively operational48 (say partition \( \pi_o \)) concerns (say for n = 3), and where “descending” partitions, say \( \pi_s, \pi_t, \pi_o \), represents coarse, medium and fine partitions, respectively.

Examples: This example only illustrates production aspects. At the strategic level one may partition a pipeline system into just one component: the entire

45Strategic management, one can claim, deals with the management of the most generic and general, year-to-year company resources: invested capital, overall market, production and service goals, etc.
46Tactical management, one can claim, deals with the management of the quarter/month-to-quarter/month resources “closest” to the implementation if strategic goals
48Operational management, one can finally claim, deals with the management of day-to-day resources “closest” to the actual market, production and services.
collection of all pipeline units, $\pi$. At the tactical level one may further partition the pipeline system into the partition of all wells, $\pi_{w}$, and the partition of all sinks, $\pi_{s}$, is a set of all routes of $\pi$ excluding wells and sinks. At the organisational level may further partition the pipeline system into the partitions of individual wells, $\pi_{w} (\pi_{w} \in \pi_{w})$, the partitions of individual sinks, $\pi_{s} (\pi_{s} \in \pi_{s})$, and the partitions of individual pipeline routes, $\pi_{r} (\pi_{r} \in \pi_{r})$.

A domain organisation serves to structure management and non-management staff levels and the allocation of strategic, tactical and operational concerns across all staff levels; and hence the “lines of command”: who does what, and who reports to whom, administratively and functionally.

Organisations are conceptual parts, that is, concepts are conceptual parts in addition, i.e., adjacent to physical parts. They serve as “place-holders” for management.

**Modelling Organisations:** We can normally model an organisation as an attribute of some, usually composite, part. Typically such a model would be in terms of the one or more partitionings of unique identifiers, $\pi : \Pi$, of domain parts, $p,P$. For example:

- **type**
  - $\text{ORG} = \text{Str} \times \text{Tac} \times \text{Ope} \times \ldots$
  - $\text{Str}, \text{Tac}, \text{Ope} = (\Pi\text{-set})\text{-set}$

- **value**
  - attr\_ORG: $P \rightarrow \text{ORG}$
  - axiom
  - $P: \text{ORG} \rightarrow \ldots \rightarrow \text{Bool}$

where we leave the details of the partitionings $\text{Str}$, $\text{Tac}$, $\text{Ope}$, ... and the axiom governing the individual partitionings and their relations for further analysis.

**Faults and Hazards:** There are erroneous and there are risky organisations. An erroneous organisation is, for example, one in which one or more partitions are left isolated with respect to there being no management “tower-holder”. A hazardous organisation is, for example, one that consists of too many partitionings, whereby related “tower-holding” management becomes confused.

### 5.6.2 Management.

By domain management we shall understand such people who (such decisions which) (i) determine, formulate and thus set standards (cf. rules and regulations, above) concerning strategic, tactical and operational decisions; (ii) who ensure that these decisions are passed on to (lower) levels of management, and to floor staff; (iii) who make sure that such orders, as they were, are indeed carried out; (iv) who handle undesirable deviations in the carrying out of these orders cum decisions; and (v) who “backstops” complaints from lower management levels and from floor staff.

**Example:** [Cf. examples on the previous page.] At the strategic level there is the overall management of the pipeline system. At the tactical level there may be the management of all wells; all sinks; specific (disjoint) routes. At the operational there may then be the management of individual wells, individual sinks, and individual groups of valves and pumps.

**Modelling Management:** Some parts are associated with strategic management. They will have their unique identifiers, $\pi : \Pi$, belong to some partition in an $\text{Str}$.$\text{Str}$. Other parts are associated with tactical management. They will have their unique identifiers, $\pi : \Pi$, belong to some partition in the corresponding $\text{Tac}$.$\text{Tac}$. Yet other parts are associated with operational management. They will have their unique identifiers, $\pi : \Pi$, belong to some partition in the corresponding $\text{Ope}$.$\text{Ope}$. The “management” parts have their attributes form corresponding states $(\sigma; \Sigma)$.

**type**
- $\Sigma_{\text{STR}}, \Sigma_{\text{TAC}}, \Sigma_{\text{OPE}}$.

An idealised rendition of management actions is:

- **value**
  - $\text{actions}\_\text{Strategic}: \Sigma_{\text{STR}} \times \Sigma_{\text{TAC}} \times \Sigma_{\text{OPE}} \rightarrow \Sigma_{\text{STR}}$
  - $\text{actions}\_\text{Tactical}: \Sigma_{\text{STR}} \times \Sigma_{\text{TAC}} \times \Sigma_{\text{OPE}} \rightarrow \Sigma_{\text{TAC}}$
  - $\text{actions}\_\text{Operational}: \Sigma_{\text{STR}} \times \Sigma_{\text{TAC}} \times \Sigma_{\text{OPE}} \rightarrow \Sigma_{\text{OPE}}$

$\text{actions}\_\text{Strategic}$ expresses that strategic management considers the “global” state $(\Sigma_{\text{STR}} \times \Sigma_{\text{TAC}} \times \Sigma_{\text{OPE}})$ but potentially changes only the “strategy” state.

$\text{actions}\_\text{Tactical}$ expresses that tactical management considers the “global” state $(\Sigma_{\text{STR}} \times \Sigma_{\text{TAC}} \times \Sigma_{\text{OPE}})$ but potentially changes only the “tactical” state.

$\text{actions}\_\text{Operational}$ expresses that tactical management considers the “global” state $(\Sigma_{\text{STR}} \times \Sigma_{\text{TAC}} \times \Sigma_{\text{OPE}})$ but potentially changes only the “operational” state.

We can normally model management as part of the behavioural model of some, usually composite part. Typically such a model would be in terms communication procedures between management, $p,P$, and their immediate subordinates, $\{p_{1},P_{1},p_{2}:P_{2},\ldots,p_{n}:P_{N}\}$: For example:

- **channel**
  - $\text{mg}: \{\{\pi,\pi\}\} | \pi_{j}:\{p_{1},p_{2},\ldots,p_{n}\} : \text{M}$
  - **value**
  - $p: \Pi \times pt:P \rightarrow \text{in, out} (\{\pi,\pi\}|\pi_{j}:\{p_{1},p_{2},\ldots,p_{n}\} : \text{M})$
  - $p(\pi,pt) \equiv \ldots$

  - $\text{let } (\pi_{j},m) = \text{query}_{\text{boss}}(p) \text{ in}$
  - $m \equiv \text{mg} \{\{\pi,\pi\}\} \text{ in} ;
  - p(\pi,\text{action}_{\text{down}}(pt,m)) \text{ end}$
  - $\text{management “listens” to staff}$

  - $\text{let } (\pi_{j},m) = \emptyset \{\text{mg}(\pi,\pi)\} \text{ in}$
  - $\text{mg}(\pi,\text{action}_{\text{down}}(pt,m)) \text{ end}$
  - $\text{management reports to boss}$

  - $\text{let } (\pi,\text{num}) = \text{query}_{\text{staff}}(pt) \text{ in}$
  - $m \equiv \text{mg}(\pi,\pi) \text{ in} ;
  - p(\pi,\text{action}_{\text{num}}(pt,m)) \text{ end}$
  - $\text{management “listens” to boss}$

  - $\text{let } (\pi,\text{num}) = \emptyset \{\text{mg}(\pi,\pi)\} \text{ in}$
  - $\text{mg}(\pi,\text{action}_{\text{up}}(pt,m)) \text{ end} \ldots$

The boss communications express that process $p$ serves a boss. All other communications express that process $p$ interacts with staff (i.e., “subordinates and “others”).

**Hazards:** [Cf. faults and hazards on the previous page.] Faults and hazards of organisations & management come about also as the result of “mis-management”: Strategic management updates tactical and operational management states. Tactical management updates strategic and operational management states. Operational management updates strategic and tactic management states. That is: these states are not clearly delineated, etcetera!
This section on organisation & management is rather terse; in fact it covers a whole, we should think, novel and interesting theory of business organisation & management.

5.7 Discussion

There may be other facets but our point has been made: that an analysis of hazards (including faults) can, we think, be beneficially structured by being related to reasonably distinct facets.

A mathematical explanation of the concept of facet is needed. One that helps partition the domain phenomena and concepts into disjoint descriptions. We are thinking about it and encourage the reader to do likewise!

6 Conclusion

The present author’s research has since the early 1970s focused on programming methodology: how to develop software such that it was correct with respect to some specification — call it requirements. The emphasis was on abstract software specifications and their refinement or transformation into code. Programming language semantics and the stage- and step-wise development of compilers, in many, up to nine stages and steps, became a highlight of the 1980s. The step from programming language semantics to domain descriptions followed: Domain descriptions, in a sense, specified the language inherent in the described domain — that is: “spoken” by its actors, etc. Since the early 1990s I therefore additionally focused on domain descriptions. Now an additional goal of software development might be achieved: securing that the software met customers’ expectations.

With the observation that requirements prescriptions can be systematically — but, of course, not automatically — “derived” from domain descriptions a bridge was established: from domains via requirements to software.

6.1 Comparison to Other Work

(Bjørner 2014b) contains a large section, Sect. 4.1 (4+ pages), which compares our domain analysis and description approach to the domain analysis approaches of Ontology and Knowledge Engineering, Database Analysis (Buchmann Diagrams, Relational Data Models, Entity Set Relations, etc., Prieto-Díaz’s work, Domain Specific Languages, Feature-oriented Domain Analysis, Software Product Line Engineering, Michael Jackson’s Problem Frames, Domain Specific Software Architectures, Domain Driven Design, Unified Modelling Language, etcetera. We refer to (Bjørner 2014b) for its lengthy discussion and almost 30 citations. (Bjørner 2014b, Sect. 4.1) shows that our approach is significantly different from the above enumerated approaches.

6.2 What Have We Achieved?

When Dr Clive Victor Boughton, on November 4, 2013, approached me on the subject of “Software Safety: New Challenges and Solutions”, I therefore, naturally questioned: can one stratify the issues of safety criticality into three phases: searching for sources of faults and hazards in domains, elaborating on these while “discovering” further sources during requirements engineering, and, finally, during early stages of software design. I believe we have answered that question partially with there being good hopes for further stratification.

Yes, I would indeed claim that we have contributed to the “greater” issues of safety critical systems by suggesting a disciplined framework for faults “discovery” and hazards: investigate separately the domains, the requirements and the design.

6.3 Further Work

But, clearly, that work has only begun.

7 Acknowledgements

I thank Dr Clive Victor Boughton of aS5Ca, ANU, for having the courage to convince his colleagues to invite me, for having inspired me to observe that faults and hazards can be “discovered” purely in the context of domain descriptions, for his support in answering my many questions, and for otherwise arranging my visit. I also, with thanks, acknowledge comments and remarks by the ASSC program chair, Dr Anthony Cant.

8 Bibliography

8.1 Notes

This conference contribution is part of a series of papers on the topic of domains. (Bjørner 2007, 2008, 2010a,b, 2011a,b, 2014b, 2009, 2010c, Bjørner & Eir 2010). In (Bjørner 2008) we show how to “derive” requirements prescriptions from domain descriptions; (Bjørner 2010a) shows techniques for describing domain facets: intrinsics, support technologies, rules & regulations, management & organisation as well as human behaviour; (Bjørner 2011b) illuminates such concepts as simulation, demos, monitoring and control in the new light afforded by the domain viewpoint; (Bjørner 2013a) speculates on various issues of “computation for humanity” (!); and, (Bjørner 2013a) relates our modelling of mereology to the classical axiom systems for mereology. (Bjørner 2014c) provides a systematic introduction to principles, techniques and tools for the analysis and description of domain endurants.

8.2 References


