Pipelines
A Domain Science & Engineering Description*

Dines Bjørner
DTU Compute, Technical University of Denmark
bjorner@gmail.com, www.imm.dtu.dk/~dibj

November 17, 2022: 14:42

Abstract

We present a description of an abstracted domain of pipelines.

The description structure follows that of the domain analysis structure into a description of endurant entities: (i) the observation of parts and fluids, (ii) the identification of unique part identifiers, (iii) the mereology of parts and (iv) the (multitude) of part and fluid attributes; and the description of perdurant entities: (v) states, (vi) channels, (vii) actions and (viii) behaviours.

The Triptych Dogma

In order to *specify* **software**,
we must understand its requirements.
In order to *prescribe* **requirements**
we must understand the **domain**.
So we must **study, analyse** and **describe** domains.

Contents

The Triptych Dogma **1**

I Opening **3**

1 **Introduction** **3**
1.1 **Domains** 4
1.2 **Domain Examples** 4
1.3 **Structure of Paper** 4

II Domain Science & Engineering **4**

*Invited paper for **FSEN 2023 : Fundamentals of Software Engineering**, Teheran, Iran, May 3–5, 2023

2 Endurants **4**
2.1 External Qualities . . . 5
2.2 Internal Qualities . . . 6
2.2.1 Unique Identifiers . . . 7
2.2.2 Mereology . . . 7
2.2.3 Attributes . . . 7
2.2.3.1 Static Attributes . . . 7
2.2.3.2 Monitorable Attributes . . . 7
2.2.3.3 Programmable Attributes . . . 7
2.2.4 Universal Qualities . . . 7
2.3 Manifest and Structure Parts . . . 7

3 Perdurants **8**
3.1 State . . . 8
3.2 Channels . . . 8
3.3 Actors . . . 8
3.4 Actions . . . 8
3.5 Events . . . 8
3.6 Behaviours . . . 8

4 Domain Analysis & Description **9**

III Example **9**

5 Endurants: External Qualities **10**
5.1 Parts . . . 10
5.2 An Endurant State . . . 11

6 Endurants: Internal Qualities **11**
6.1 Unique Identification . . . 12
6.2 Mereology . . . 12
6.2.1 PLS Mereology . . . 12
6.2.2 Unit Mereologies . . . 13
6.3 Pipeline Concepts, I . . . 13
6.3.1 Pipe Routes . . . 13
6.3.2 Well-formed Routes . . . 14
6.3.3 Embedded Routes . . . 15
6.3.4 A Theorem . . . 15
6.3.5 Fluids . . . 16
6.4 Attributes . . . 16
6.4.1 Unit Flow Attributes . . . 16
6.4.2 Unit Metrics . . . 18
6.4.3 Wellformed Unit Metrics . . . 19
6.4.4 Summary . . . 19
6.4.5 Fluid Attributes . . . 20
6.4.6 Pipeline System Attributes . . . 21
6.5 Pipeline Concepts, II: Flow Laws . . . 21

7 Perdurants **22**
7.1 State . . . 22
7.2 Channel . . . 22
7.3 Actions . . . 22
7.4 Behaviours . . . 23
7.4.1 Behaviour Kinds . . . 23
7.4.2 Behaviour Signatures . . . 23
7.4.2.1 Behaviour Definitions . . . 24
7.4.2.2 The Pipeline System Behaviour . . . 24
7.4.2.3 The Pump Behaviours . . . 25
7.4.2.4 The Valve Behaviours . . . 26
7.4.3 Sampling Monitorable Attribute Values . . . 26
7.4.4 System Initialisation . . . 27

IV Summarizing **27**

8 Conclusion **27**
8.1 Software Development . . . 28
8.1.1 Domain Facets . . . 28
8.1.2 Software Requirements . . . 28
8.1.3 Software Design . . . 28
8.2 The R&D of A Full Scale, Realistic Pipeline System Domain . . . 29
8.3 A Composite Challenge . . . 29
8.4 Acknowledgments . . . 29

9 References **29**

V Appendices **31**

A Indexes **31**

B Illustrations of Pipeline Phenomena **34**

Part I
Opening

1 Introduction

We present a torso of a model of the general concept of pipelines. For illustrations of pipeline phenomena please refer to Appendix B. The methodology espoused in this paper is documented in [13]. This paper is a drastic reformulation of a report[1] worked out in connection with PhD lectures at TU Graz, the Technical University of Graz, Austria, in the fall of 2008.

[1] `http://www.imm.dtu.dk/~dibj/2022/tehran/tugraz-oil.pdf`: We invite the interested reader to take a more than cursory glance at this, Dec. 16, 2008, report, say after the study of the current paper!

1.1 Domains

By a *domain* we shall understand a *rationally describable* segment of a *discrete dynamics* segment of a *human assisted reality*, i.e., of the world; its *solid or fluid entities*: *natural* ["God-given"] and *artefactual* ["man-made"], and its *living species entities*: *plants* and *animals* – including, notably, *humans* ■

The definition relies on the understanding of the terms *'rationally describable'*, *'discrete dynamics'*, *'human assisted'*, *'solid'* and *'fluid'*. The last two will be explained later. By *rationally describable* we mean that what is described can be understood, including reasoned about, in a rational, that is, logical manner. By *discrete dynamics* we imply that we shall basically, in this primer, rule out such domain phenomena which have properties which are continuous with respect to their time-wise, i.e., dynamic, behaviour. By *human-assisted* we mean that the domains – that we are interested in modelling – have, as an important property, that they possess man-made entities.

1.2 Domain Examples

Example domains are:

- **Rivers –** with their natural sources, deltas, tributaries, waterfalls, etc., and their man-made dams, harbours, locks, etc. [14]
- **Road nets –** with street segments and intersections, traffic lights, and automobiles.
- **Pipelines –** with their wells, pipes, valves, pumps, forks, joins and wells [11].
- **Container terminals –** with their container vessels, containers, cranes, trucks, etc. [12] ■

1.3 Structure of Paper

This paper is mostly that of an example. Bearing in mind that the SEN2023 Conference at which it is to be presented takes place in Tehran, Iran, I chose the example to be that of a sketch of a generic pipeline domain: oil or gas. This introductory section has very briefly outlined elements of the domain analysis & description method treated in depth in [13]. I hope my local audience will follow suit !

Part II
Domain Science & Engineering

In this part we shall cover, basically, an ontology for analysing & describing domains.

Ontology is the branch of metaphysics dealing with the nature of being, that is, a set of concepts and categories in a subject area or domain that shows their properties and the relations between them.

2 Endurants

Endurants are those quantities of domains that we can observe (see and touch), in *space*, as "complete" entities at no matter which point in *time* – "material" entities that persists,

endures.

2.1 External Qualities

of endurants of a manifest domain are, in a simplifying sense, those we can see, touch and have spatial extent. They, so to speak, take form.

We shall analyse and describe external qualities according to the following ontology.

0 Universes of discourse consist of **non-describable phenomena** [1] and **describable phenomena** [2].

1 Non-describable phenomena will here be left further un-analysed.

2 Describable phenomena, are also called **entities** [3].

3 Entities are either **endurants** [4] or **perdurants** [16].

4 Endurants are either **solid** (solids are separate, individual or distinct in form or concept, or, rephrasing: have 'body' [or magnitude] of three-dimensions: length, breadth and depth [17, Vol. II, pg. 2046]) [5], or **fluid** (fluids are prolonged, without interruption, in an unbroken series or pattern; or, rephrasing: a substance (liquid, gas or plasma) having the property of flowing, consisting of particles that move among themselves [17, Vol. I, pg. 774])) [7].

5 Solids are either **parts** [6] or **living species** [11].

6 Parts are either **atomic** [8] or **compounds** [9].

7 Fluids are presently left further un-analysed.

8 Atomic parts are presently left further un-analysed.

9 Compounds are either **Cartesians** (a Cartesian has a given number of endurants) or **part sets** (a part set has a possibly varying number of endurants) [10].

10 Part sets consist of an indefinite set of endurants of either the same sort, or distinct, different sorts.

11 Living species are either **animals** [12] or **plants** [13].

12 Plants are here left further un-analysed.

13 Animals are either **humans** [14] or **other ...** [15].

14 Humans are here left further un-analysed.

15 Other ... is here left further un-analysed.

16 Perdurants are either **instantaneous** (an instantaneous perdurant occurs at a (or any) single point in time and manifests itself in a similarly instantaneous state change – where a state is the internal qualities value of any assembly of endurants.) [17] or **prolonged** (a Prolonged Perdurant occurs over time, perdures for either an indefinite or an infinite time interval) [20].

17 Instantaneous perdurants are either **actions** (an action is an internally provoked instantaneous state change) [18] or **events** (an event is an externally provoked instantaneous state change.) [19].

18 We shall here leave actions further un-analysed.

19 We shall here leave events further un-analysed.

19 We shall here rename prolonged perdurants into **behaviours** (a behaviour is a set of sequences of actions, events and behaviours.)

20 Behaviours are here left further un-analysed.

We refer to Fig. 1. The above 20 point enumeration corresponds to a top-down, breadth-first traversal of, first the endurants, then the perdurant, sub-trees of Fig. 1.

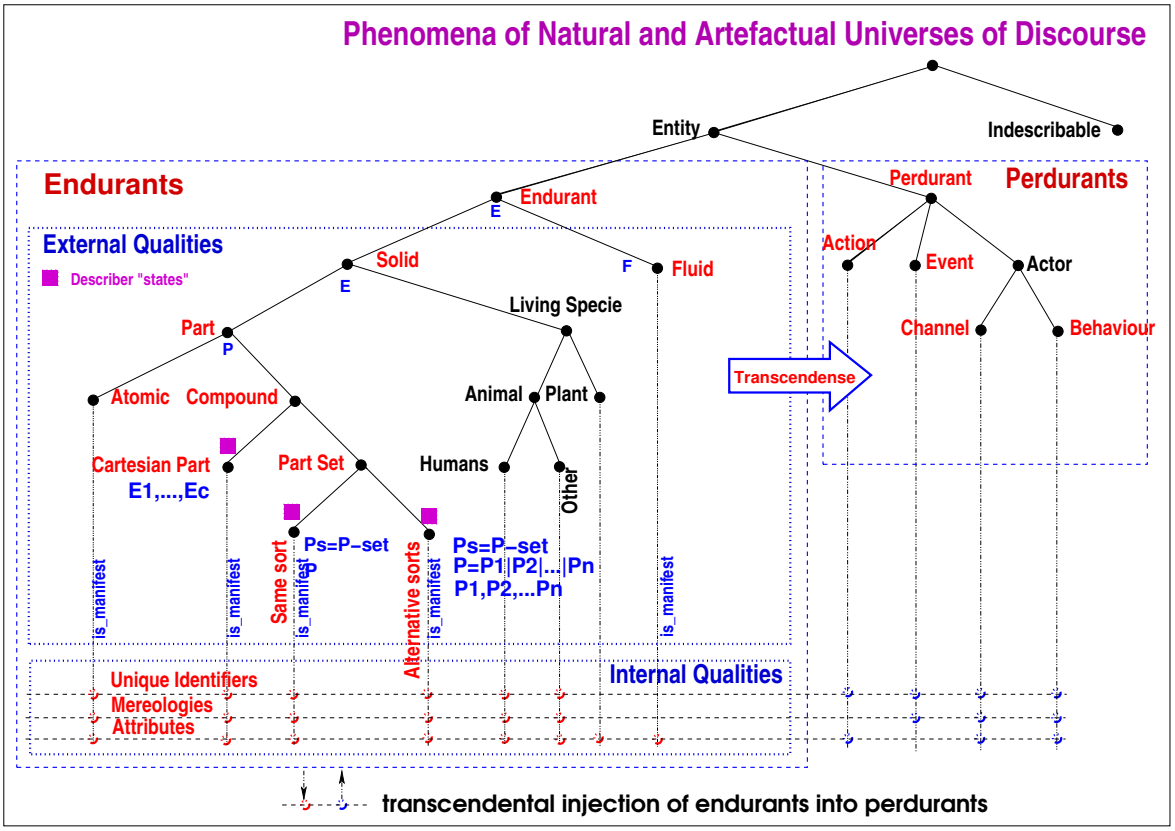

Figure 1: Upper Ontology

2.2 Internal Qualities

are those properties [of endurants] that do not occupy *space* but can be measured or spoken about.

We shall divide our analysis & description of internal qualities into three concerns: unique identification, mereologies and attributes.

2.2.1 Unique Identifiers

Unique identity is an immaterial property that distinguishes two *spatially* distinct solids.

2.2.2 Mereology

Mereology is a theory of [endurant] part-hood relations: of the relations of an [endurant] parts to a whole and the relations of [endurant] parts to [endurant] parts within that whole.

2.2.3 Attributes

Attributes are those properties of endurants that are not *spatially* observable, but can be either physically (electronically, chemically, or otherwise) measured or can be objectively spoken about.

With Michael A. Jackson [15] we distinguish between three kinds of attributes.

2.2.3.1 Static Attributes [21] **static attributes:** being endurant attributes whose values are constant.

2.2.3.2 Monitorable Attributes [22] **monitorable attributes:** being endurant attributes whose values can be "sampled", and for a subset, the **biddable attributes**, can also be set, by some actor, but when sampled may not exactly be the value that was [just] set.

2.2.3.3 Programmable Attributes [23] **programmable attributes:** being endurant attributes whose values can be set, and will "keep" that value until next 'set'.

2.2.4 Universal Qualities

are those properties of a domain which holds universally of any domain! We list the following as universal qualities: **space**, **time** and **intentionality**.

Space and *Time* can be argued to be concepts that arise by *transcendental deduction* from logical necessities, i.e., can rationally be reasoned to hold of any domain in any universe.

Intentionality[2] "expresses" conceptual, abstract relations between otherwise, or seemingly unrelated entities.

In this paper we shall not illustrate examples of temporality nor intentionality.

2.3 Manifest and Structure Parts

We shall distinguish between *manifest* and *structure* parts. A manifest part is one which to which we shall [later] ascribe internal qualities.[3] A structure part is one to which we shall **not** ascribe internal qualities. Structure parts has the pragmatic purpose to decompose domain analyses and descriptions into manageable textst. Thus

[2] The Oxford English Dictionary [17] characterises intentionality as follows: *"the quality of mental states (e.g. thoughts, beliefs, desires, hopes) which consists in their being directed towards some object or state of affairs"*.

[3] – and much later, in Sect. 7, transcendentally deduce into behaviours!

[22] there is a predicate: is_manifest, applicable to endurants, and

[23] there is a predicate: is_structure, likewise applicable to endurants.

[24] If one yields **true** the other yields **false**, and vice versa.

value
[25] is_manifest: E → **Bool**
[26] is_structure: E → **Bool**
axiom
[27] ∀ e:E • is_manifest(e) ≡ ∼is_structure(e)

3 Perdurants

Perdurants are those quantities of domains for which only a fragment exists, in *space*, if we look at or touch them at any given snapshot in *time*.

We shall analyse perdurants into a number of concepts.

3.1 State

There is a concept of **state** – here taken to be a[ny] set of *manifest endurants*.

3.2 Channels

There is a concept of **channel** – a means for behaviours (see next) to synchronize & communicate, i.e., interact.

3.3 Actors

There is a concept of **actor** – a means for sustaining actions, events and behaviours.

3.4 Actions

There is a concept of **action** – a means for effecting [orderly] state changes;

3.5 Events

There is a concept of **event**s – causing surreptitious, i.e., not-planned-for, state changes;

3.6 Behaviours

And there is a concept of **behaviour**s – syntactically speaking, being sets of sequences of actions, events and behaviours.

By transcendental deduction we shall "morph" parts into behaviours.

We shall focus on the behaviours of manifest parts.

Behaviours, mathematically speaking, are functions.

As functions they take arguments and "deliver" result values.

The arguments are decomposed into four kinds: (i) the unique identifier of the part being transcendentally deduced into a behaviour; (ii) the mereology of that part; (iii) the static attribute values of that part; (iv) the monitorable attributes of that part; (v) the programmable attribute values of that part; and (vi) the identification of the channels over which the behaviour interacts with other, mereologically "connected" behaviours.

The **behaviour signature** consists of the name of the behaviour and the function type expression over these arguments and the type of the result values. If the behaviour is indefinite, i.e., 'goes on forever", that result type is the **Unit**[4] type expression; else, if the behaviour does yield a explicit values, then the type [expression] for those values.

• • •

4 Domain Analysis & Description

The method being touted in [13] suggests the following phases and steps to be undertaken when modelling a domain:

- Endurants
  - External Qualities
    - Discovery of solids and fluids
  - Internal Qualities, part by part:
    - Unique Identifiers;
    - Mereologies;
    - Attributes;
    - Intentional Pull[5]
- Perdurants
  - States
  - Channels
  - Actions
  - Behaviours
    - Signatures
    - Definitions
    - Initialisation

We refer to Fig. 1. Section 2 (External Qualities) corresponds to the traversal of the left dashed line box of Fig. 1. Section 2.2 (Internal Qualities) corresponds to the traversal of left the vertical and horisontal dashed lines of Fig. 1. Section 3 (Perdurants) corresponds to the traversal of the right dashed line box and of the vertical and horisontal dashed lines Fig. 1.

Part III
Example

As this paper is to be presented, in Iran, at a conference organised by **IPM**, the Iranian institute for Research in Fundamental Sciences, previously **I**nstitute for Studies in Theoretical **P**hysics and **M**athematics, its author thought it appropriate to encourage **IPM** to work out a mathematical model for pipelines – an Iranian specialty ! I refer to Sect. 8.3, the Conclusion, for more on this challenge !

Descriptions alternate between enumerated narratives and formalisations. The formalisations alternative between defining types, observer functions, predicates, axioms, functions,

[4] **Unit** stands for the state-to-state changing function value, designated by ().
[5] – not covered in this paper !

etc. An Appendix index, Sect. A (Pages 31–34), may help the reader around the very many formulas.

5 Endurants: External Qualities

We follow the ontology of Fig. 2, the lefthand dashed box labelled *External Qualities*.

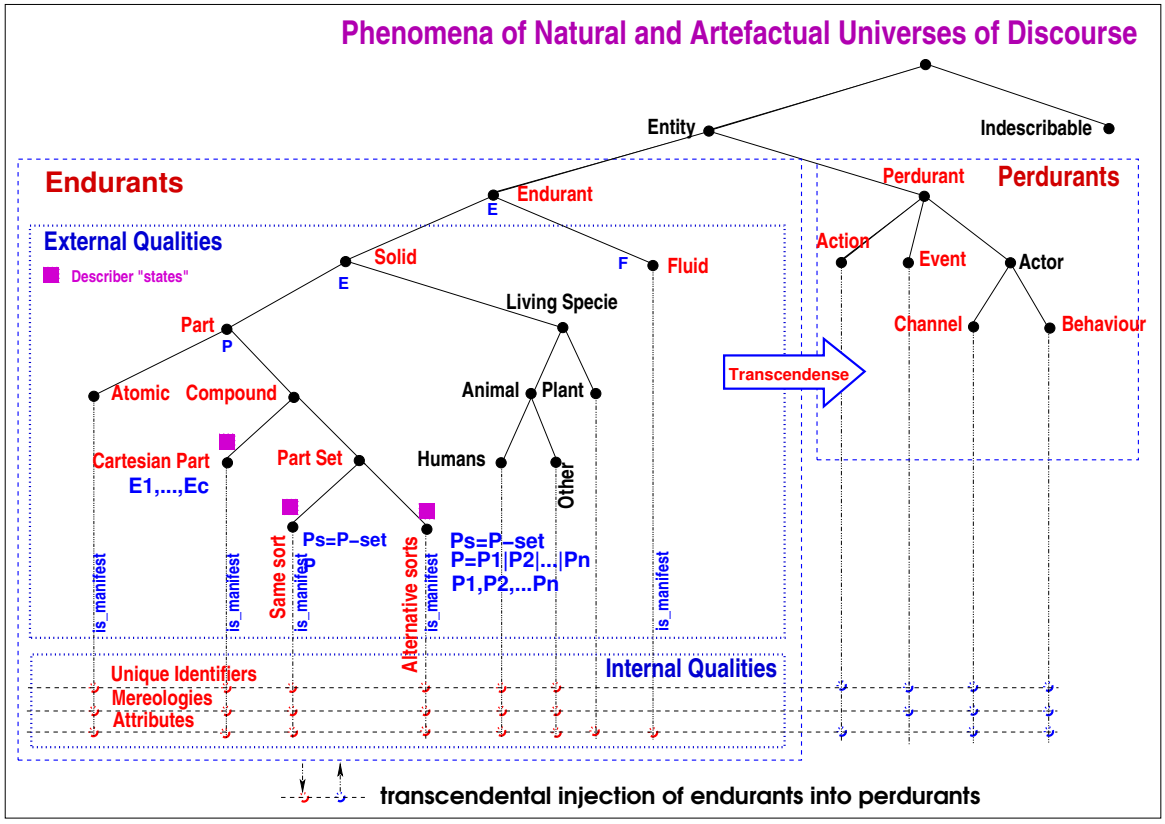

Figure 2: Upper Ontology

5.1 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. 6.2) and the routing of its pipes (cf. Sect. 6.3.2).
3. A pipeline unit is either a well, a pipe, a pump, a valve, a fork, a join, a plate[6], 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.

[6] A *plate* unit is a usually circular, flat steel plate used to "begin" or "end" a pipe segment.

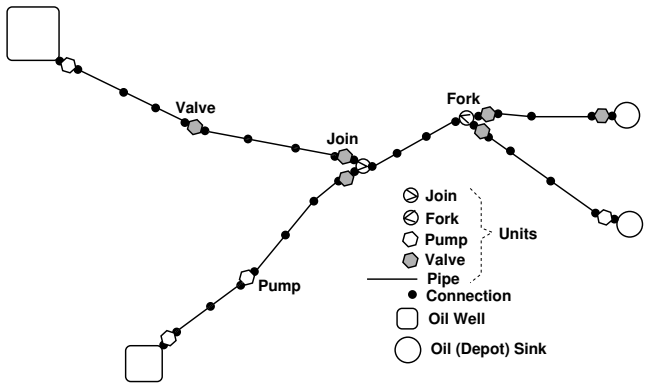

Figure 3: An example pipeline system

type
1. PLS′, U, M
2. PLS = {| pls:PLS′•wf_PLS(pls) |}

value
2. wf_PLS: PLS → **Bool**
2. wf_PLS(pls) ≡
2. wf_Mereology(pls)∧wf_Routes(pls)∧wf_Metrics(pls)[7]
1. obs_Us: PLS → U-**set**
1. obs_M: PLS → M

type
3. U = We | Pi | Pu | Va | Fo | Jo | Pl | Si
4. We :: Well
4. Pi :: Pipe
4. Pu :: Pump
4. Va :: Valv
4. Fo :: Fork
4. Jo :: Join
4. Pl :: Plate
4. Si :: Sink

5.2 An Endurant State

5. For a given pipeline system
6. we exemplify an endurant state σ
7. composed of the given pipeline system and all its manifest units, i.e., without plates.

value
5. pls:PLS

variable
6. σ := collect_state(pls)

value
7. collect_state: PLS
7. collect_state(pls) ≡ {pls} ∪ obs_Us(pls) \ Pl

6 Endurants: Internal Qualities

We follow the ontology of Fig. 2 on the preceding page, the lefthand vertical and horisontal lines.

[7] wf_Mereology, wf_Routes and wf_Metrics will be explained in Sects. 6.2.2 on page 13, 6.3.2 on page 15, and 6.4.3 on page 19.

6.1 Unique Identification

8. The pipeline system, as such,
9. has a unique identifier, distinct (different) from its pipeline unit identifiers.
10. Each pipeline unit is uniquely distinguished by its unit identifier.
11. There is a state of all unique identifiers.

```
type
9.   PLSI
10.  UI
value
8.   pls:PLS
9.   uid_PLS: PLS → PLSI
10.  uid_U: U → UI
variable
11.  σ_uid := { uid_PLS(pls) } ∪ xtr_UIs(pls)
axiom
10.  ∀ u,u′:U•{u,u′}⊆obs_Us(pls)⇒(u≠u′⇒uid_UI(u)≠uid_UI(u′))
10.  ∧ uid_PLS(pls) ∉ {uid_UI(u)|u:U•u ∈ obs_Us(pls)}
```

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

```
value
12.  xtr_UIs: PLS → UI-set
12.  xtr_UIs(pls) ≡ {uid_UI(u)|u:U•u ∈ obs_Us(pls)}
```

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

```
theorem:
13.  ∀ pls:PLS•card obs_Us(pl)=card xtr_UIs(pls)
```

6.2 Mereology

6.2.1 PLS Mereology

14. The mereology of a pipeline system is the set of unique identifiers of all the units of that system.

```
type
14.  PLS_Mer = UI-set
value
14.  mereo_PLS: PLS → PLS_Mer
axiom
14.  ∀ uis:PLS_Mer • uis = card xtr_UIs(pls)
```

6.2.2 Unit Mereologies

15. Each unit is connected to zero, one or two other existing input units and zero, one or two other existing output units as follows:
    a. A well unit is connected to exactly one output unit (and, hence, has no "input").
    b. A pipe unit is connected to exactly one input unit and one output unit.
    c. A pump unit is connected to exactly one input unit and one output unit.
    d. A valve is connected to exactly one input unit and one output unit.
    e. A fork is connected to exactly one input unit and two distinct output units.
    f. A join is connected to exactly two distinct input units and one output unit.
    g. A plate is connected to exactly one unit.
    h. A sink is connected to exactly one input unit (and, hence, has no "output").

```
type
15.  MER = UI-set × UI-set
value
15.  mereo_U: U → MER
axiom
15.  wf_Mereology: PLS → Bool
15.  wf_Mereology(pls) ≡
15.    ∀ u:U•u ∈ obs_Us(pls)⇒
15.      let (iuis,ouis) = mereo_U(u) in iuis ∪ ouis ⊆ xtr_UIs(pls) ∧
15.        case (u,(card uius,card ouis)) of
15a..          (mk_We(we),(0,1)) → true,
15b..          (mk_Pi(pi),(1,1)) → true,
15c..          (mk_Pu(pu),(1,1)) → true,
15d..          (mk_Va(va),(1,1)) → true,
15e..          (mk_Fo(fo),(1,1)) → true,
15f..          (mk_Jo(jo),(1,1)) → true,
15f..          (mk_PI(pl),(0,1)) → true, "begin"
15f..          (mk_PI(pl),(1,0)) → true, "end"
15h..          (mk_Si(si),(1,1)) → true,
15.          _ → false end end
```

6.3 Pipeline Concepts, I

6.3.1 Pipe Routes

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

```
type
16.  R′ = U^ω
```

```
16.   R = {| r:Route′•wf_Route(r) |}
17.   RD = UI^ω
      axiom
17.   ∀ rd:RD • ∃ r:R•rd=descriptor(r)
      value
17.   descriptor: R → RD
17.   descriptor(r) ≡ ⟨uid_UI(r[i])|i:Nat•1≤i≤len r⟩
```

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

```
      value
18.   adjacent: U × U → Bool
18.   adjacent(u,u′) ≡ let (,ouis)=mereo_U(u),(iuis,)=mereo_U(u′) in ouis ∩ iuis ≠ {} end
```

19. Given a pipeline system, *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 *pls*.
    b. Let u, u' be any units of *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 *pls*.
    c. If r and r' are routes of *pls* such that the last element of r is the same as the first element of r', then $r^\frown\mathbf{tl}r'$ is a route of *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 19a.–19c..

```
      value
19.   Routes: PLS → RD-infset
19.   Routes(pls) ≡
19a..       let rs = ⟨⟩ ∪
19b..             {⟨uid_UI(u),uid_UI(u′)⟩|u,u′:U•{u,u′}⊆obs_Us(pls) ∧ adjacent(u,u′)}
19c..           ∪ {r^tl r′|r,r′:R•{r,r′}⊆rs}
19d..       in rs end
```

6.3.2 Well-formed Routes

20. A route is acyclic if no two route positions reveal the same unique unit identifier.

```
      value
20.   is_acyclic_Route: R → Bool
20.   is_acyclic_Route(r) ≡ ∼∃ i,j:Nat•{i,j}⊆inds r ∧ i≠j ∧ r[i]=r[j]
```

21. 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
21. wf_Routes: PLS → Bool
21. wf_Routes(pls) ≡
21.     non_circular(pls) ∧ are_embedded_Routes(pls)

21. is_non_circular_PLS: PLS → Bool
21. is_non_circular_PLS(pls) ≡
21.     ∀ r:R•r ∈ routes(p)∧acyclic_Route(r)
```

22. 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
22. well_to_sink_Routes: PLS → R-set
22. well_to_sink_Routes(pls) ≡
22.   let rs = Routes(pls) in
22.   {r|r:R•r ∈ rs ∧ is_We(r[1]) ∧ is_Si(r[len r])} end
```

23. A pipeline system is well-formed if all of its routes are embedded in well-to-sink routes.

```
23. are_embedded_Routes: PLS → Bool
23. are_embedded_Routes(pls) ≡
23.     let wsrs = well_to_sink_Routes(pls) in
23.     ∀ r:R • r ∈ Routes(pls) ⇒
23.         ∃ r′:R,i,j:Nat •
23.           r′ ∈ wsrs
23.         ∧ {i,j}⊆inds r′∧i≤j
23.         ∧ r = ⟨r′[k]|k:Nat•i≤k≤j⟩ end
```

6.3.3 Embedded Routes

24. For every route we can define the set of all its embedded routes.

```
value
24. embedded_Routes: R → R-set
24. embedded_Routes(r) ≡ {⟨r[k]|k:Nat•i≤k≤j⟩ | i,j:Nat• i {i,j}⊆inds(r) ∧ i≤j}
```

6.3.4 A Theorem

25. 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

```
    theorem:
25.  ∀ pls:PLS •
25.  let rs = Routes(pls),
25.      wsrs = well_to_sink_Routes(pls) in
25a..  rs =
25b..      wsrs ∪
25c..      ∪ {{r′|r′:R • r′ ∈ is_embedded_Routes(r″)} | r″:R • r″ ∈ wsrs}
24.  end
```

6.3.5 Fluids

26. The only fluid of concern to pipelines is the gas[8] or liquid[9] which the pipes transport[10].

```
    type
26.     GoL [ = M ]
    value
26.     obs_GoL: U → GoL
```

6.4 Attributes

6.4.1 Unit Flow Attributes

27. A number of attribute types characterise units:
    a. estimated current well capacity (barrels of oil, etc.),
    b. pump height (a static attribute),
    c. current pump status (not pumping, pumping; a programmable attribute),
    d. current valve status (closed, open; a programmable attribute) and
    e. flow (barrels/second, a biddable attribute).

```
    type
27a..    WellCap
27b..    Pump_Height
27c..    Pump_State == {|not_pumping,pumping|}
27d..    Valve_State == {|closed,open|}
27e..    Flow
```

28. Flows can be added and subtracted,
29. added distributively and
30. flows can be compared.

[8] Gaseous materials include: air, gas, etc.
[9] Liquid materials include water, oil, etc.
[10] The description of this document is relevant only to gas or oil pipelines.

value

28. $\oplus, \ominus$: Flow×Flow → Flow
29. $\oplus$: Flow-**set** → Flow
30. $<, \leq, =, \neq, \geq, >$: Flow × Flow → **Bool**

31. Properties of pipeline units include
    a. estimated current well capacity (barrels of oil, etc.) [a biddable attribute],
    b. pipe length [a static attribute],
    c. current pump height [a biddable attribute],
    d. current valve open/close status [a programmable attribute],
    e. current [$\mathcal{L}$aminar] in-flow at unit input [a monitorable attribute],
    f. current $\mathcal{L}$aminar] in-flow leak at unit input [a monitorable attribute],
    g. maximum [$\mathcal{L}$aminar] guaranteed in-flow leak at unit input [a static attribute],
    h. current [$\mathcal{L}$aminar] leak unit interior [a monitorable attribute],
    i. current [$\mathcal{L}$aminar] flow in unit interior [a monitorable attribute],
    j. maximum $\mathcal{L}$aminar] guaranteed flow in unit interior [a monitorable attribute],
    k. current [$\mathcal{L}$aminar] out-flow at unit output [a monitorable attribute],
    l. current [$\mathcal{L}$aminar] out-flow leak at unit output [a monitorable attribute] and
    m. maximum guaranteed $\mathcal{L}$aminar out-flow leak at unit output [a static attribute.

type
31e. In_Flow = Flow
31f. In_Leak = Flow
31g. Max_In_Leak = Flow
31h. Body_Flow = Flow
31i. Body_Leak = Flow
31j. Max_Flow = Flow
31k. Out_Flow = Flow
31l. Out_Leak = Flow
31m. Max_Out_Leak = Flow
value
31a. attr_WellCap: We → WellCap
31b. attr_LEN: Pi → LEN
31c. attr_Height: Pu → Height
31d. attr_ValSta: Va → VaSta
31e. attr_In_Flow: U → UI → Flow
31f. attr_In_Leak: U → UI → Flow
31g. attr_Max_In_Leak: U → UI → Flow
31h. attr_Body_Flow: U → Flow
31i. attr_Body_Leak: U → Flow
31j. attr_Max_Flow: U → Flow
31k. attr_Out_Flow: U → UI → Flow
31l. attr_Out_Leak: U → UI → Flow
31m. attr_Max_Out_Leak: U → UI → Flow

32. Summarising we can define a two notions of flow:
    a. static and
    b. monitorable.

type
32a. Sta_Flows = Max_In_Leak×In_Max_Flow>Max_Out_Leak
32b. Mon_Flows = In_Flow×In_Leak×Body_Flow×Body_Leak×Out_Flow×Out_Leak

6.4.2 Unit Metrics

Pipelines are laid out in the terrain. Units have length and diameters. Units are positioned in space: have altitude, longitude and latitude positions of its one, two or three connection PoinTs[11].

33. length (a static attribute),
34. diameter (a static attribute) and
35. position (a static attribute).

type
33. LEN
34. ◯
35. POS == mk_One(pt:PT) | mk_Two(ipt:PT,opt:PT)
35. | mk_OneTwo(ipt:PT,opts:(lpt:PT,rpt:PT))
35. | mk_TwoOne(ipts:(lpt:PT,rpt:PT),opt:PT)
35. PT = Alt × Lon × Lat
35. Alt, Lon, Lat = ...

value
33. attr_LEN: U → LEN
34. attr_◯: U → ◯
35. attr_POS: U → POS

We can summarise the metric attributes:

36. Units are subject to either of four (mutually exclusive) metrics:
    a. Length, diameter and a one point position.
    b. Length, diameter and a two points position.
    c. Length, diameter and a one+two points position.
    d. Length, diameter and a two+one points position.

type
36. Unit_Sta = Sta1_Metric | Sta2_Metric | Sta12_Metric | Sta21_Metric
36a. Sta1_Metric = LEN × Ø × mk_One(pt:PT)
36b. Sta2_Metric = LEN × Ø × mk_Two(ipt:PT,opt:PT)
36c. Sta12_Metric = LEN × Ø × mk_OneTwo(ipt:PT,opts:(lpt:PT,rpt:PT))
36d. Sta21_Metric = LEN × Ø × mk_TwpOne(ipts:(lpt:PT,rpt:PT),opt:PT)

[11] 1 for *well*s, *plate*s and *sink*s; 2 for *pipe*s, *pump*s and *valve*s; 1+2 for *fork*s, 2+1 for *join*s.

6.4.3 Wellformed Unit Metrics

The points positions of neighbouring units must "fit" one-another.

37. Without going into details we can define a predicate, wf_Metrics, that applies to a pipeline system and yields **true** iff neighbouring units must "fit" one-another.

value
37. wf_Metrics: PLS → **Bool**
37. wf_Metrics(pls) ≡ ...

6.4.4 Summary

We summarise the static, monitorable and programmable attributes for each manifest part of the pipeline system:

type
PLS_Sta = PLS_net×...
PLS_Mon = ...
PLS_Prg = PLS_Σ×...
Well_Sta = Sta1_Metric×Sta_Flows×Orig_Cap×...
Well_Mon = Mon_Flows×Well_Cap×...
Well_Prg = ...
Pipe_Sta = Sta2_Metric×Sta_Flows×LEN×...
Pipe_Mon = Mon_Flows×In_Temp×Out_Temp×...
Pipe_Prg = ...
Pump_Sta = Sta2_Metric×Sta_Flows×Pump_Height×...
Pump_Mon = Mon_Flows×...
Pump_Prg = Pump_State×...
Valve_Sta = Sta2_Metric×Sta_Flows×...
Valve_Mon = Mon_Flows×In_Temp×Out_Temp×...
Valve_Prg = Valve_State×...
Fork_Sta = Sta12_Metric×Sta_Flows×...
Fork_Mon = Mon_Flows×In_Temp×Out_Temp×...
Fork_Prg = ...
Join_Sta = Sta21_Metric×Sta_Flows×...
Join_Mon = Mon_Flows×In_Temp×Out_Temp×...
Join_Prg = ...
Sink_Sta = Sta1_Metric×Sta_Flows×Max_Vol×...
Sink_Mon = Mon_Flows×Curr_Vol×In_Temp×Out_Temp×...
Sink_Prg = ...

38. Corresponding to the above three attribute categories we can define "collective" attribute observers:

value
38. sta_A_We: We → Sta1_Metric×Sta_Flows×Orig_Cap×...

38. mon_A_We: We → ηMon_Flows×ηWell_Cap×ηIn_Temp×ηOut_Temp×...
38. prg_A_We: We → ...
38. sta_A_Pi: Pi → Sta2_Metric×Sta_Flows×LEN×...
38. mon_A_Pi: Pi → $\mathcal{N}$Mon_Flows×ηIn_Temp×ηOut_Temp×...
38. prg_A_Pi: Pi → ...
38. sta_A_Pu: Pu → Sta2_Metric×Sta_Flows×LEN×...
38. mon_A_Pu: Pu → $\mathcal{N}$Mon_Flows×ηIn_Temp×ηOut_Temp×...
38. prg_A_Pu: Pu → Pump_State×...
38. sta_A_Va: Va → Sta2_Metric×Sta_Flows×LEN×...
38. mon_A_Va: Va → $\mathcal{N}$Mon_Flows×ηIn_Temp×ηOut_Temp×...
38. prg_A_Va: Va → Valve_State×...
38. sta_A_Fo: Fo → Sta12_Metric×Sta_Flows×...
38. mon_A_Fo: Fo → $\mathcal{N}$Mon_Flows×ηIn_Temp×ηOut_Temp×...
38. prg_A_Fo: Fo → ...
38. sta_A_Jo: Jo → Sta21_Metric×Sta_Flows×...
38. mon_A_Jo: Jo → Mon_Flows×ηIn_Temp×ηOut_Temp×...
38. prg_A_Jo: Jo → ...
38. sta_A_Si: Si → Sta1_Metric×Sta_Flows×Max_Vol×...
38. mon_A_Si: Si → $\mathcal{N}$Mon_Flows×ηIn_Temp×ηOut_Temp×...
38. prg_A_Si: Si → ...

38. $\mathcal{N}$Mon_Flows ≡ (ηIn_Flow,ηIn_Leak,ηBody_Flow,ηBody_Leak,ηOut_Flow,ηOut_Leak)

Monitored flow attributes are [to be] passed as arguments to behaviours *by reference* so that their monitorable attribute values can be sampled.

6.4.5 Fluid Attributes

Fluids, we here assume, oil, as it appears in the pipeline units have no unique identity, have not mereology, but does have attributes: hydrocarbons consisting predominantly of aliphatic, alicyclic and aromatic hydrocarbons. It may also contain small amounts of nitrogen, oxygen, and sulfur compounds

39. We shall simplify, just for illustration, crude oil fluid of units to have these attributes:
    a. volume,
    b. viscosity,
    c. temperature,
    d. paraffin content (%age),
    e. naphtenes content (%age),

type
39. Oil
39a.. Vol
39b.. Visc
39c.. Temp
39d.. Paraffin
39e.. Naphtene
value
39b.. obs_Oil: U → Oil
39a.. attr_Vol: Oil → Vol

39b.. attr_Visc: Oil → Visc
39c.. attr_Temp: Oil → Temp
39d.. attr_Paraffin: Oil → Paraffin
39e.. attr_Naphtene: Oil → Naphtene

6.4.6 Pipeline System Attributes

The "root" pipeline system is a compound. In its transcendentally deduced behavioral form it is, amongst other "tasks", entrusted with the monitoring and control of all its units. To do so it must, as a basically static attribute possess awareness, say in the form of a net diagram of how these units are interconnected, together with all their internal qualities, by type and by value. Next we shall give a very simplified account of the possible pipeline system attribute.

40. We shall make use, in this example, of just a simple pipeline state, pls_ω.

The pipeline state, pls_ω, embodies all the information that is relevant to the monitoring and control of an entire pipeline system, whether static or dynamic.

type
40. PLS_Ω

6.5 Pipeline Concepts, II: Flow Laws

41. "What flows in, flows out !". For $\mathcal{L}$aminar flows: for any non-well and non-sink unit the sums of input leaks and in-flows equals the sums of unit and output leaks and out-flows.

Law:
41. ∀ u:U\We\Si •
41. sum_in_leaks(u) ⊕ sum_in_flows(u) =
41. attr_body_Leak$_{\mathcal{L}}$(u) ⊕
41. sum_out_leaks(u) ⊕ sum_out_flows(u)

value
sum_in_leaks: U → Flow
sum_in_leaks(u) ≡ **let** (iuis,) = mereo_U(u) **in** ⊕ {attr_In_Leak$_{\mathcal{L}}$(u)(ui)|ui:UI•ui ∈ iuis} **end**
sum_in_flows: U → Flow
sum_in_flows(u) ≡ **let** (iuis,) = mereo_U(u) **in** ⊕ {attr_In_Flow$_{\mathcal{L}}$(u)(ui)|ui:UI•ui ∈ iuis} **end**
sum_out_leaks: U → Flow
sum_out_leaks(u) ≡ **let** (,ouis) = mereo_U(u) **in** ⊕ {attr_Out_Leak$_{\mathcal{L}}$(u)(ui)|ui:UI•ui ∈ ouis} **end**
sum_out_flows: U → Flow
sum_out_flows(u) ≡ **let** (,ouis) = mereo_U(u) **in** ⊕ {attr_Out_Leak$_{\mathcal{L}}$(u)(ui)|ui:UI•ui ∈ ouis} **end**

42. "What flows out, flows in !". For $\mathcal{L}$aminar 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:
42. ∀ u,u′:U•adjacent(u,u′) ⇒
42. **let** (,ouis)=mereo_U(u), (iuis′,)=mereo_U(u′) **in**
42. **assert:** uid_U(u′) ∈ ouis ∧ uid_U(u) ∈ iuis′
42. attr_Out_Flow$_{\mathcal{L}}$(u)(uid_U(u′)) =
42. attr_In_Leak$_{\mathcal{L}}$(u)(uid_U(u))⊕attr_In_Flow$_{\mathcal{L}}$(u′)(uid_U(u)) **end**

These "laws" should hold for a pipeline system without plates.

7 Perdurants

We follow the ontology of Fig. 2 on page 10, the right-hand dashed box labeled *Perdurants* and the right-hand vertical and horisontal lines.

7.1 State

We introduce concepts of *manifest* and *structure* endurants. The former are such compound endurants (Cartesians of sets) to which we ascribe internal qualities; the latter are such compound endurants (Cartesians of sets) to which we **do not** ascribe internal qualities. The distinction is pragmatic.

43. For any given pipeline system we suggest the state to consist of the manifest endurants of all its non-plate units.

value
43. σ = obs_Us(pls)

7.2 Channel

44. There is a [global] array channel indexed by a "set pair" of distinct manifest endurant part identifiers – signifying the possibility of the syncharonisation and communication between any pair of pipeline units and between these and the pipeline system, cf. last, i.e., bottom-most diagram of Fig. 13 on page 36.

channel
44. { ch[{i,j}] | {i,j}:(PLSI|UI) • {i,j}⊆σ_{id} }

7.3 Actions

These are, informally, some of the actions of a pipeline system:

45. **start pumping**: from a state of not pumping to a state of pumping "at full blast!".[12]
46. **stop pumping**: from a state of (full) pumping to a state of no pumping at all.

[12] – that is, we simplify, just for the sake of illustration, and do not consider "intermediate" states of pumping.

47. **open valve**: from a state of a fully closed valve to a state of fully open valve.[13]

48. **close valve**: from a state of a fully opened valve to a state of fully closed valve.

We shall not define these actions in this paper. But they will be referred to in the *pipeline_system* (Items 67a., 67b., 67c.), the *pump* (Items 70a., 70b.) and the *valve* (Items 73a., 73b.) behaviours.

7.4 Behaviours

7.4.1 Behaviour Kinds

There are eight kinds of behaviours:

49. the pipeline system behaviour;[14]
50. the [generic] well behaviour,
51. the [generic] pipe behaviour,
52. the [generic] pump behaviour,
53. the [generic] valve behaviour,
54. the [generic] fork behaviour,
55. the [generic] join behaviour,
56. the [generic] sink behaviour.

7.4.2 Behaviour Signatures

57. The *pipeline_system* behaviour, *pls*,

58. The *well* behaviour signature lists the unique well identifier, the well mereology, the static well attributes, the monitorable well attributes, the programmable well attributes and the channels over which the well [may] interact with the pipeline system and a pipeline unit.

59. The *pipe* behaviour signature lists the unique pipe identifier, the pipe mereology, the static pipe attributes, the monitorable pipe attributes, the programmable pipe attributes and the channels over which the pipe [may] interact with the pipeline system and its two neighbouring pipeline units.

60. The *pump* behaviour signature lists the unique pump identifier, the pump mereology, the static pump attributes, the monitorable pump attributes, the programmable pump attributes and the channels over which the pump [may] interact with the pipeline system and its two neighbouring pipeline units.

61. The *valve* behaviour signature lists the unique valve identifier, the valve mereology, the static valve attributes, the monitorable valve attributes, the programmable valve attributes and the channels over which the valve [may] interact with the pipeline system and its two neighbouring pipeline units.

[13] – cf. Footnote 12 on the facing page.

[14] This "PLS" behaviour summarises the either global, i.e., *SCADA*[15]-like behaviour, or the fully distributed, for example, manual, human-operated behaviour of the monitoring and control of the entire pipeline system.

[15] Supervisory Control And Data Acquisition

62. The *fork* behaviour signature lists the unique fork identifier, the fork mereology, the static fork attributes, the monitorable fork attributes, the programmable fork attributes and the channels over which the fork [may] interact with the pipeline system and its three neighbouring pipeline units.
63. The *join* behaviour signature lists the unique join identifier, the join mereology, the static join attributes, the monitorable join attributes, the programmable join attributes and the channels over which the join [may] interact with the pipeline system and its three neighbouring pipeline units.
64. The *sink* behaviour signature lists the unique sink identifier, the sink mereology, the static sing attributes, the monitorable sing attributes, the programmable sink attributes and the channels over which the sink [may] interact with the pipeline system and its one or more pipeline units.

value
57. pls: plso:PLSI → pls_mer:PLS_Mer → PLS_Sta → PLS_Mon →
57. PLS_Prg → { ch[{plsi,ui}] | ui:UI • ui ∈ σ_{ui} } **Unit**
58. well: wid:WI → well_mer:MER → Well_Sta → Well_mon →
58. Well_Prgr → { ch[{plsi,ui}] | wi:WI • ui ∈ σ_{ui} } **Unit**
59. πipe: UI → pipe_mer:MER → Pipe_Sta → Pipe_mon →
59. Pipe_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σ_{ui} } **Unit**
60. pump: pi:UI → pump_mer:MER → Pump_Sta → Pump_Mon →
60. Pump_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σ_{ui} } **Unit**
61. valve: vi:UI → valve_mer:MER → Valve_Sta → Valve_Mon →
61. Valve_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σ_{ui} } **Unit**
62. fork: fi:FI → fork_mer:MER → Fork_Sta → Fork_Mon →
62. Fork_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σ_{ui} } **Unit**
63. join: ji:JI → join_mer:MER → Join_Sta → Join_Mon →
63. Join_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σ_{ui} } **Unit**
64. sink: si:SI → sink_mer:MER → Sink_Sta → Sink_Mon →
64. Sink_Prgr → { ch[{plsi,ui}] | ui:UI • ui ∈ σ_{ui} } **Unit**

7.4.2.1 Behaviour Definitions

We show the definition of only three behaviours:

- the **pipe_line_system** behaviour,
- the **pump** behaviour and
- the **valve** behaviour.

7.4.2.2 The Pipeline System Behaviour

65. The pipeline system behaviour
66. calculates, based on its programmable state, its next move;
67. if that move is [to be] an action on a named

a. pump, whether to start or stop pumping, then the named pump is so informed, whereupon the pipeline system behaviour resumes in the new pipeline state; or

b. valve, whether to open or close the valve, then the named valve is so informed, whereupon the pipeline system behaviour resumes in the new pipeline state; or

c. unit, to collect its monitorable attribute values for monitoring, whereupon the pipeline system behaviour resumes in the further updated pipeline state;

d. et cetera;

```
value
65.  pls(plsi)(uis)(pls_msta)(pls_mon)(pls_ω) ≡
66.    let (to_do,pls_ω′) = calculate_next_move(plsi,pls_mer,pls_msta,pls_mon,pls_prgr) in
67.    case to_do of
67a.     mk_Pump(pi,α) →
67a.           ch[{plsi,pi}] ! α assert: α ∈ {stop_pumping,pump};
67a.           pls(plsi)(pls_mer)(pls_msta)(pls_mon)(pls_ω′),
67b.     mk_Valve(vi,α) →
67b.           ch[{plsi,vi}] ! α assert: α ∈ {open_valve,close_valve};
67b.           pls(plsi)(pls_mer)(pls_msta)(pls_mon)(pls_ω′),
67c.     mk_Unit(ui,monitor) →
67c.           ch[{plsi,ui}] ! monitor;
67c.           pls(plsi)(pls_mer)(pls_msta)(pls_mon)(update_pls_ω(ch[{plsi,ui}] ?,ui)(pls_ω′)),
67d.     ... end
65     end
```

We leave it to the reader to define the calculate_next_move function!

7.4.2.3 The Pump Behaviours

68. The [generic] pump behaviour internal non-deterministically alternates between

69. doing own work (...), or

70. accepting pump directives from the pipeline behaviour.

    a. If the directive is either to start or stop pumping, then that is what happens – whereupon the pump behaviour resumes in the new pumping state.

    b. If the directive requests the values of all monitorable attributes, then these are *gathered*, communicated to the pipeline system behaviour – whereupon the pump behaviour resumes in the "old" state.

```
value
68.  pump(π)(pump_mer)(pump_sta)(pump_mon)(pump_prgr) ≡
69.    ...
70.    ⌈⌉ let α = ch[{plsi,π}] ? in
70.      case α of
70a..          stop_pumping ∨ pump
70a..              → pump(π)(pump_mer)(pump_sta)(pump_mon)(α)[16]end,
```

```
70b..          monitor
70b..              → let mvs = gather_monitorable_values(π,pump_mon) in
70b..                ch[ {plsi,π} ] ! mvs;
70b..                pump(π)(pump_mer)(pump_sta)(pump_mon)(pump_prgr) end
70.        end
```

We leave it to the reader to defined the gather_monitorable_values function.

7.4.2.4 The Valve Behaviours

71. The [generic] valve behaviour internal non-deterministically alternates between
72. doing own work (...), or
73. accepting valve directives from the pipeline system.
    a. If the directive is either to open or close the valve, then that is what happens – whereupon the pump behaviour resumes in the new valve state.
    b. If the directive requests the values of all monitorable attributes, then these are *gathered*, communicated to the pipeline system behaviour – whereupon the valve behaviour resumes in the "old" state.

```
value
71.  valve(vi)(valv_mer)(valv_sta)(valv_mon)(valv_prgr) ≡
72.    ...
73.    ⌈⌉ let α = ch[ {plsi,π} ] ? in
73.       case α of
73a..          open_valve ∨ close_valve
73a..              → valve(vi)(val_mer)(val_sta)(val_mon)(α)[17]end,
73b..          monitor
73b..              → let mvs = gather_monitorable_values(vi,val_mon) in
73b..                ch[ {plsi,π} ] ! (vi,mvs);
73b..                valve(vi)(val_mer)(val_sta)(val_mon)(val_prgr) end
73.       end
```

7.4.3 Sampling Monitorable Attribute Values

Static and programmable attributes are, as we have seen, *passed by value* to behaviours. Monitorable attributes "surreptitiously" change their values so, as a technical point, these are *passed by reference* – by *passing attribute type names*.

74. From the name, ηA, of a monitorable attribute and the unique identifier, u_i, of the part having the named monitorable attribute one can then, "dynamically", "on-the-fly", as the part behaviour "moves-on", retrieve the value of the monitorable attribute. This can be illustrated as follows:

[16] Updating the programmable pump state to either **stop_pumping** or **pump** shall here be understood to mean that the pump is set to not pump, respectively to pump.

[17] Updating the programmable valve state to either **open_valve** or **close_valve** shall here be understood to mean that the valve is set to open, respectively to closed position.

75. The unique identifier u_i is used in order to retrieve, from the global parts state, σ, that identified part, p.
76. Then attr_A is applied to p.

value
74. retr_U: UI → Σ → U
74. retr_U(ui)(σ) ≡ **let** u:U • u ∈ σ∧uid_U(u)=ui **in** u **end**
75. retr_AttrVal: UI × ηA → Σ → A
76. retr_AttrVal(ui)(ηA)(σ) ≡ attr_A(retr_U(ui)(σ))

retr_AttrVal(...)(...)(...) can now be applied in the body of the behaviour definitions, for example in gather_monitorable_values.

7.4.4 System Initialisation

System initialisation means to "morph" all manifest parts into their respective behaviours, initialising them with their respective attribute values.

77. The *pipeline system* behaviour is initialised and "put" in parallel with the parallel compositions of
78. all initialised *well*,
79. all initialised *pipe*,
80. all initialised *pump*,
81. all initialised *valve*,
82. all initialised *fork*,
83. all initialised *join* and
84. all initialised *sink* behaviours.[18]

value
77. pls(uid_PLS(pls))(mereo_PLS(pls))((pls))((pls))((pls))
78. ‖ ‖ { well(uid_U(we))(mereo_U(we))(sta_A_We(we))(mon_A_We(we))(prg_A_We(we)) | we:Well • w ∈ σ }
79. ‖ ‖ { pipe(uid_U(pi))(mereo_U(pi))(sta_A_Pi(pi))(mon_A_Pi(pi))(prg_A_Pi(pi)) | pi:Pi • pi ∈ σ }
80. ‖ ‖ { pump(uid_U(pu))(mereo_U(pu))(sta_A_Pu(pu))(mon_A_Pu(pu))(prg_A_Pu(pu)) | pu:Pump • pu ∈ σ }
81. ‖ ‖ { valv(uid_U(va))(mereo_U(va))(sta_A_Va(va))(mon_A_Va(va))(prg_A_Va(va)) | va:Well • va ∈ σ }
82. ‖ ‖ { fork(uid_U(fo))(mereo_U(fo))(sta_A_Fo(fo))(mon_A_Fo(fo))(prg_A_Fo(fo)) | fo:Fork • fo ∈ σ }
83. ‖ ‖ { join(uid_U(jo))(mereo_U(jo))(sta_A_Jo(jo))(mon_A_J(jo))(prg_A_J(jo)) | jo:Join • jo ∈ σ }
84. ‖ ‖ { sink(uid_U(si))(mereo_U(si))(sta_A_Si(si))(mon_A_Si(si))(prg_A_Si(si)) | si:Sink • si ∈ σ }

The sta_..., mon_..., and prg_A... functions are defined in Items 38 on page 19.
Note: ‖ { f(u)(...) | u:U • u ∈ {} } ≡ **()**.

Part IV
Summarizing

8 Conclusion

We have, in Part I (Sects. 2–3) briefly sketched a method for analysing & describing artefactual domains, and in Part III (Sects. 5–7) sketched a model for a conceptual domain of pipeline

[18] Plates are treated as are structures, i.e., not "behaviourised" !

systems. We must emphasize, however, that the author, me, of this example really does not know much about pipelines. He would like to. He hopes that perhaps publishing this paper may put him in contact with oil pipeline professionals who might enlighten him ?!

It has been hinted at in this paper, but it need be made quite clear. The basis for the domain analysis & description methodology outlined in Part I (Pages 4–9) owes much to the Philosophy of Kai Sørlander [20, 21, 22, 23, 24].

8.1 Software Development

8.1.1 Domain Facets

We have dealt, in this paper, with the core method of domain engineering and illustrated one application. There is, however, more to domain engineering. The notion of **domain facets** is treated in [13, *Chapter 8*]. 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. Example facets are: *intrinsics, support technologies, rules and regulations, scripts, license languages, management & organisation and human behaviour.*

8.1.2 Software Requirements

From domain descriptions one can, methodologically develop requirements for software. By software requirements (cf., IEEE Standard 610.12) we shall understand *"A condition or capability needed by a user to solve a problem or achieve an objective"*. The requirements aim at a *machine* By a machine we shall understand the hardware and software that is to be designed and which are to satisfy the requirements. [13, *Chapter 9*] outlines a method for requirements development. It endows the "classical" form of requirements engineering with a structured set of development stages and steps: (a) first a *domain requirements* stage[19], (b) to be followed by an *interface requirements* stage[20], and (c) to be concluded by a *machine requirements* stage[21]; (3) it further structures and gives a reasonably precise contents to the stage of domain requirements: (i) first a *projection* step, (ii) then an *instantiation* step, (iii) then a *determination* step, (iv) then an *extension* step, and (v) finally a *fitting* step with these five steps possibly being iterated; and (4) it also structures and gives a reasonably precise contents to the stage of interface requirements based on a notion of shared entities. Each of the steps (iv) open for the possibility of simplifications. Steps (ac) and (i-v), we claim, are new. They reflect a serious contribution, we claim, to a logical structuring of the field of requirements engineering and its very many otherwise seemingly diverse concerns.

8.1.3 Software Design

Finally there is the software design & coding phase. In the context of the *Triptych* approach, one which is based on a combination of narrative and formal expressions, we refer to [2, 3, 4].

[19] By *domain requirements* we understand such requirements which can be expressed using terms sôly of the domain.

[20] By *interface requirements* we understand such requirements which can be expressed using terms both of the domain and the machine.

[21] By *machine requirements* we understand such requirements which can be expressed using terms sôly of the the machine.

8.2 The R&D of A Full Scale, Realistic Pipeline System Domain

We refer to [13] concluding chapter's Sect. 11.5, Pages 316–317: **On How to Conduct a Domain Analysis & Description Project.** We emphasize that any such project, for pipeline systems, necessarily has a significant research element: I am, in particular, thinking of the PDE modelling of the dynamic flows withing a pipeline system.

8.3 A Composite Challenge

The challenges are these:

- (i) To define the continuous, laminar and turbulent, flows for each kind of pipeline unit.
- (ii) To define the flows for an entire pipeline system, i.e., for the flow within the net of all units.
- (iii) To research the interface between the discrete mathematics (logic, set theory, etc.) of, in this case, RSL, and the continuous mathematics of PDEs etc.

Items (i–ii) is expected to involve the use of *Bernoulli*[22] and *Navier-Stokes*[23] partial differential equations (PDEs) as well as integrals. Item (i) should be "fairly" easy! Item (ii) seems "tricky": To compose, for all possible well-formed configurations of units, the PDEs for constituent units. I think that this is a new kind of mathematical modelling challenge.

We refer to documents related to the above: [1, 16, 19, 18]

8.4 Acknowledgments

I wish, with this my first paper for an Iranian conference to pay tribute to the late Prof. Lotfi Zadeh, my mentor in the years 1970–1972. Zadeh "led me away" from IBM Research "back" into academia. He invited me to give lectures on *Programming Language Semantics* at UC Berkeley, California, 1970–1972. My affection for Lotfi Zadeh carried over, somehow, to Prof. Ahmad R. Sharafat, former director of ITRC, Iranian Telecommunication Research Center – and a student of mine at UC Berkeley in 1971. Prof. Sharafat invited me to visit Iran. So I enjoyed three such previous visits: 1991, 1993 and 1999. Fondest thanks to both!

9 References

[1] Nagia Mohamed Dafaa Allah. *Solving a Mathematical Model For Gas Flow in Pipelines*. Phd thesis, Faculty of Mathematical Sciences, University of Khartoum, JG5R+968, Barlaman Ave, Khartoum, Sudan, October 2008. 124 Pages, https://core.ac.uk/download/pdf/71671586.pdf. Cited on page 29.

[2] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. See [5, 8]. Cited on page 28.

[3] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen. See [6, 9]. Cited on page 28.

[22] https://www.mathematik.ch/mathematiker/daniel_bernoulli.html
[23] https://en.wikipedia.org/wiki/Navier%E2%80%93Stokes_equations

[4] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. See [7, 10]. Cited on page 28.

[5] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Qinghua University Press, 2008. Cited on page 29.

[6] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Qinghua University Press, 2008. Cited on page 29.

[7] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Qinghua University Press, 2008. Cited on page 30.

[8] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 1: Abstraction and Modelling*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010. Cited on page 29.

[9] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 2: Specification of Systems and Languages*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010. Cited on page 29.

[10] Dines Bjørner. **Chinese:** *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Qinghua University Press. Translated by Dr Liu Bo Chao et al., 2010. Cited on page 30.

[11] Dines Bjørner. Pipelines – a Domain www.imm.dtu.dk/~dibj/pipe-p.pdf. Experimental Research Report 2013-2, DTU Compute and Fredsvej 11, DK-2840 Holte, Denmark, Spring 2013. Cited on page 4.

[12] Dines Bjørner. Container Terminals. www.imm.dtu.dk/ dibj/2018/yangshan/maersk-pa.pdf. Technical report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, September 2018. An incomplete draft report; currently 60+ pages. Cited on page 4.

[13] Dines Bjørner. *Domain Science & Engineering – A Foundation for Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, 2021. Cited on pages 3, 4, 9, 28, and 29.

[14] Dines Bjørner. Rivers and Canals. www.imm.dtu.dk/~dibj/2021/Graphs/Rivers-and-Canals.pdf. Technical Report, Technical University of Denmark, Fredsvej 11, DK-2840 Holte, Denmark, March 2021. Cited on page 4.

[15] Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley, Reading, England, 1995. Cited on page 7.

[16] LiangSun. Mathematical modeling of the flow in a pipeline with a leak. *Mathematics and Computers in Simulation*, 82(11):2253–2267, July 2012. Cited on page 29.

[17] W. Little, H.W. Fowler, J. Coulson, and C.T. Onions. *The Shorter Oxford English Dictionary on Historical Principles*. Clarendon Press, Oxford, England, 1973, 1987. Two vols. Cited on pages 5 and 7.

[18] Maple. Pipeline Design & Analysis – Maple Flow Examples. The Internet. `https://www.maplesoft.com/products/MapleFlow/Pipeline-Design-Software/examples.aspx`. Cited on page 29.

[19] *Mathematical modelling of pipelines, including equipment, levelling sharp changes in fluid pressure*, volume IOP Conference Series Materials Science and Engineering, January 2021. DOI:10.1088/1757-899X/1030/1/012149: `https://www.researchgate.net/publication/348520073_Mathematical_modelling_of_pipelines_including_equipment_levelling_sharp_changes_in_fluid_pressure`. Cited on page 29.

[20] Kai Sørlander. *Det Uomgængelige – Filosofiske Deduktioner [The Inevitable – Philosophical Deductions, with a foreword by Georg Henrik von Wright]*. Munksgaard · Rosinante, Copenhagen, Denmark, 1994. 168 pages. Cited on page 28.

[21] Kai Sørlander. *Under Evighedens Synsvinkel [Under the viewpoint of eternity]*. Munksgaard · Rosinante, Copenhagen, Denmark, 1997. 200 pages. Cited on page 28.

[22] Kai Sørlander. *Den Endegyldige Sandhed [The Final Truth]*. Rosinante, Copenhagen, Denmark, 2002. 187 pages. Cited on page 28.

[23] Kai Sørlander. *Indføring i Filosofien [Introduction to The Philosophy]*. Informations Forlag, Copenhagen, Denmark, 2016. 233 pages. Cited on page 28.

[24] Kai Sørlander. *Den rene fornufts struktur [The Structure of Pure Reason]*. Ellekær, Slagelse, Denmark, 2022. Cited on page 28.

Part V
Appendices

A Indexes

Concepts:, 5
- Action, 6, 8, 22
- Actor, 8
- Animal, 5
- Atomic, 5
- Attribute
  - Monitorable, 7
  - Programmable, 7
  - Static, 7
- Attributes, 7
- Behaviour, 6, 8, 23
  - Definitions, 24
  - Signature, 9, 23
- Cartesian, 5
- Channel, 8, 22
- Compound, 5
- Definitions
  - Behaviour, 24
- Domain, 4
  - Analysis & Description, 9
  - Facet, 28
  - Requirements, 28
- Endurants, 4, 10
- Event, 6, 8
- External Qualities, 5
- Facet
  - Domain, 28
- Human, 5
- INTENT, 7
- Interface
  - Requirements, 28
- Internal Qualities, 6
- Living Species, 5
- Machine
  - Requirements, 28
- Mereology, 7
- Monitorable Attribute, 7
- Ontology, 4
- Part, 5
- Part Sets, 5
- Parts, 10
- Perdurants, 8, 22
- Plant, 5
- Programmable Attribute, 7
- Requirement
  - Software, 28
- Requirements
  - Domain, fn. 19, 28
  - Interface, fn. 20, 28
  - Machine, fn. 21, 28
- Signature
  - Behaviour, 23
- Software
  - Design, 28
  - Requirement, 28
- SPACE, 7
- State, 8, 22
- Static Attribute, 7
- TIME, 7
- Transcendental Deduction, 7
- Triptych Dogma, 1
- Unique Identity, 7

All Formulas:
- $< \iota 30$, 17
- $= \iota 30$, 17
- $> \iota 30$, 17
- $\bigcirc \iota 34$, 18
- $\geq \iota 30$, 17
- $\leq \iota 30$, 17

⊖ ι28, 17
⊕ ι28, 17
⊕ ι29, 17
σ ι6, 11
σ ι43, 22
σ_{uid} ι8, 12
≠ ι30, 17
adjacent ι18, 14
Alt ι35, 18
are_ embedded_ Routes ι23, 15
attr_ ◯ ι34, 18
attr_ Body_ Flow ι31h., 17
attr_ Body_ Leak ι31i., 17
attr_ In_ Flow ι31e., 17
attr_ In_ Leak ι31f., 17
attr_ LEN ι33, 18
attr_ Max_ Flow ι31j., 17
attr_ Max_ In_ Leak ι31g., 17
attr_ Max_ Out_ Leak ι31m., 17
attr_ Out_ Flow ι31k., 17
attr_ Out_ Leak ι31l., 17
attr_ POS ι35, 18
Body_ Flow ι31h., 17
Body_ Leak ι31i., 17
ch ι44, 22
collect_ state ι7, 11
descriptor ι17, 14
embedded_ Routes ι24, 15
Flow ι27e., 16
Fo ι4, 11
fork ι62, 24
GoL ι26, 16
In_ Flow ι31e., 17
In_ Flow≡Out_ Flow ι41, 21
In_ Leak ι31f., 17
initialisation ι77–84, 27
is_ acyclic_ Route ι20, 14
is_ animal [11], 5
is_ atomic [6], 5
is_ Cartesian [9], 5
is_ compound [6], 5
is_ endurant [4], 5
is_ entity [2], 5
is_ fluid [4], 5
is_ human [13], 5
is_ living_ species [5], 5
is_ manifest [25], 8
is_ monitorable_ attribute [22], 7
is_ non_ circular_ PLS ι21, 15
is_ part [5], 5
is_ part_ set [9], 5
is_ perdurant [4], 5
is_ plant [11], 5
is_ programmable_ attribute [23], 7
is_ solid [5], 5
is_ static_ attribute [21], 7
is_ structure [26], 8
Jo ι4, 11
join ι63, 24
Lat ι35, 18
LEN ι33, 18
Lon ι35, 18
M ι1, 11
Max_ Flow ι31j., 17
Max_ In_ Leak ι31g., 17
Max_ Out_ Leak ι31m., 17
MER ι15, 13
mereo_ PLS ι14, 12
mereo_ U ι15, 13
Mon_ Flows ι32b., 17
obs_ GoL ι26, 16
obs_ M ι1, 11
obs_ Us ι1, 11
Out_ Flow ι31k., 17
Out_ Flow≡In_ Flow ι41, 22
Out_ Leak ι31l., 17
Pi ι4, 11
pipe ι59, 24
Pl ι4, 11
pls ι65, 25
pls ι5, 11
pls ι57, 24
PLS ι2, 11
PLS′ ι1, 11
PLS_ Mer ι14, 12
PLSI ι9, 12
POS ι35, 18
PT ι35, 18
Pu ι4, 11
pump ι68, 25
pump ι60, 24
Pump_ Height ι27b., 16
Pump_ State ι27c., 16
R ι16, 14
R′ ι16, 13
RD ι17, 14
retr_ AttrVal ι75, 27
retr_ U ι74, 27
Route Describability ι17, 14
Routes ι19, 14
Routes of a PLS ι25, 16
Si ι4, 11
sink ι64, 24
Sta12_ Metric ι36c., 18
Sta1_ Metric ι36a., 18
Sta21_ Metric ι36d., 18
Sta2_ Metric ι36b., 18
Sta_ Flows ι32a., 17
U ι1, 11
U ι3, 11
UI ι10, 12
uid_ PLS ι9, 12
uid_ U ι10, 12
Unique Endurants ι13, 12
Unique Identification ι10, 12
Unit_ Sta ι36, 18
Va ι4, 11
valve ι71, 26
valve ι61, 24
Valve_ State ι27d., 16
We ι4, 11
well ι58, 24
well_ to_ sink_ Routes ι22, 15
WellCap ι27a., 16
Wellformed Mereologies ι14, 12
wf_ Mereology ι15, 13
wf_ Metrics ι37, 19
wf_ PLS ι2, 11
wf_ Routes ι21, 15
xtr_ UIs ι12, 12

Types:
PLS_ Mer ι14a, 12
Wellformed Mereologies ι14a, 12

Types
Endurant:
Fo ι4a, 11
GoL ι26a, 16
Jo ι4a, 11
M ι1a, 11
Pi ι4a, 11
Pl ι4a, 11
PLS ι2a, 11
PLS′ ι1a, 11
Pu ι4a, 11
Si ι4a, 11
U ι1a, 11
U ι3a, 11
Va ι4a, 11
We ι4a, 11
Unique identifier:
PLSI ι9a, 12
UI ι10a, 12
Mereology:
MER ι15a, 13
Attribute:
◯ ι34a, 18
Alt ι35a, 18
Body_ Flow ι31h.a, 17
Body_ Leak ι31i.a, 17
Flow ι27e.a, 16
In_ Flow ι31e.a, 17
In_ Leak ι31f.a, 17
Lat ι35a, 18
LEN ι33a, 18
Lon ι35a, 18
Max_ Flow ι31j.a, 17

Max_ In_ Leak ι31g.a, 17
Max_ Out_ Leak ι31m.a, 17
Mon_ Flows ι32b.a, 17
Out_ Flow ι31k.a, 17
Out_ Leak ι31l.a, 17
POS ι35a, 18
PT ι35a, 18
Pump_ Height ι27b.a, 16
Pump_ State ι27c.a, 16
Sta12_ Metric ι36c.a, 18
Sta1_ Metric ι36a.a, 18
Sta21_ Metric ι36d.a, 18
Sta2_ Metric ι36b.a, 18
Sta_ Flows ι32a.a, 17
Unit_ Sta ι36a, 18
Valve_ State ι27d.a, 16
WellCap ι27a.a, 16

Other types:
R ι16a, 14
R′ ι16a, 13
RD ι17a, 14

Values:
pls ι5, 11

Functions:
adjacent ι18, 14
collect_ state ι7, 11
descriptor ι17, 14
embedded_ Routes ι24, 15
retr_ AttrVal ι75, 27
retr_ U ι74, 27
Routes ι19, 14
well_ to_ sink_ Routes ι22, 15
xtr_ UIs ι12, 12

Operations:
$<$ ι30, 17
$=$ ι30, 17
$>$ ι30, 17
$\geq$ ι30, 17
$\leq$ ι30, 17
$\ominus$ ι28, 17
$\oplus$ ι28, 17
$\oplus$ ι29, 17
$\neq$ ι30, 17

Observers:
attr_ $\bigcirc$ ι34, 18
attr_ Body_ Flow ι31h., 17
attr_ Body_ Leak ι31i., 17
attr_ In_ Flow ι31e., 17
attr_ In_ Leak ι31f., 17
attr_ LEN ι33, 18
attr_ Max_ Flow ι31j., 17
attr_ Max_ In_ Leak ι31g., 17
attr_ Max_ Out_ Leak ι31m., 17
attr_ Out_ Flow ι31k., 17
attr_ Out_ Leak ι31l., 17
attr_ POS ι35, 18
mereo_ PLS ι14, 12
mereo_ U ι15, 13
obs_ GoL ι26, 16
obs_ M ι1, 11
obs_ Us ι1, 11
uid_ PLS ι9, 12
uid_ U ι10, 12

Predicates:
are_ embedded_ Routes ι23, 15
is_ acyclic_ Route ι20, 14
is_ animal [11], 5
is_ atomic [6], 5
is_ Cartesian [9], 5
is_ compound [6], 5
is_ endurant [4], 5
is_ entity [2], 5
is_ fluid [4], 5
is_ human [13], 5
is_ living_ species [5], 5
is_ manifest [25], 8
is_ monitorable_ attribute [22], 7
is_ part [5], 5
is_ part_ set [9], 5
is_ perdurant [4], 5
is_ plant [11], 5
is_ programmable_ attribute [23], 7
is_ solid [5], 5
is_ static_ attribute [21], 7
is_ structure [26], 8

States:
σ ι6, 11
σ ι43, 22
σ_{uid} ι8, 12

Axioms:
Route Describability ι17, 14
Unique Identification ι10, 12

Well-formedness:
is_ non_ circular_ PLS ι21, 15
wf_ Mereology ι15, 13
wf_ Metrics ι37, 19
wf_ PLS ι2, 11
wf_ Routes ι21, 15

Channel:
ch ι44, 22

Behaviour

Signatures:
fork ι62, 24
join ι63, 24
pipe ι59, 24
pls ι57, 24
pump ι60, 24
sink ι64, 24
valve ι61, 24
well ι58, 24

Definitions:
pls ι65, 25
pump ι68, 25
valve ι71, 26

Initialisation:
initialisation ι77–84, 27

Theorems:
Routes of a PLS ι25, 16
Unique Endurants ι13, 12

Laws:
In_ Flow≡Out_ Flow ι41, 21
Out_ Flow≡In_ Flow ι41, 22

B Illustrations of Pipeline Phenomena

Figure 4: **The Planned Nabucco Pipeline:** http://en.wikipedia.org/wiki/Nabucco_Pipeline

Figure 5: **Pipeline Construction**

Figure 6: **Pipe Segments**

Figure 7: **Valves**

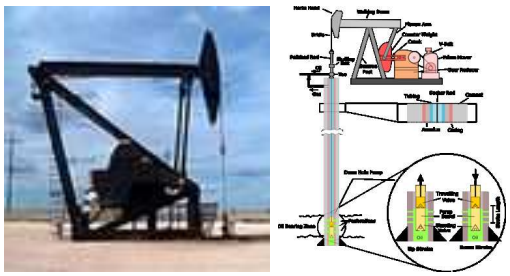

Figure 8: **Oil Pumps**

Figure 9: **Gas Compressors**

Figure 10: **New and Old Pigs**

Figure 11: **Pig Launcher, Receiver**

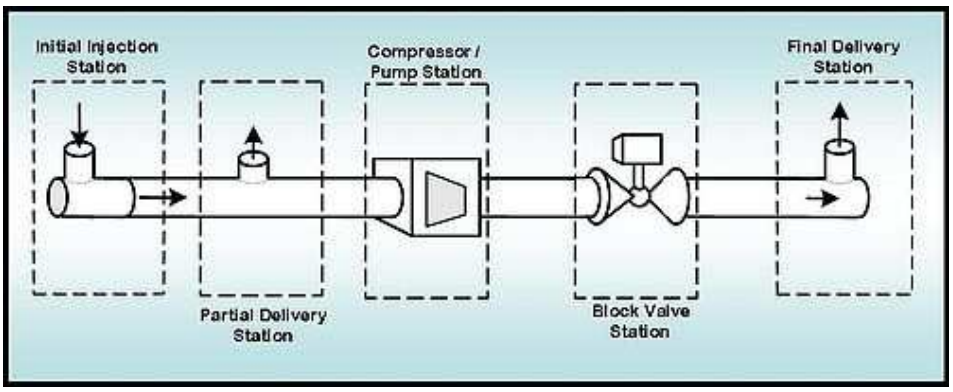

Figure 12: **Leftmost: A Well. 2nd from left: a Fork. Rightmost: a Sink**

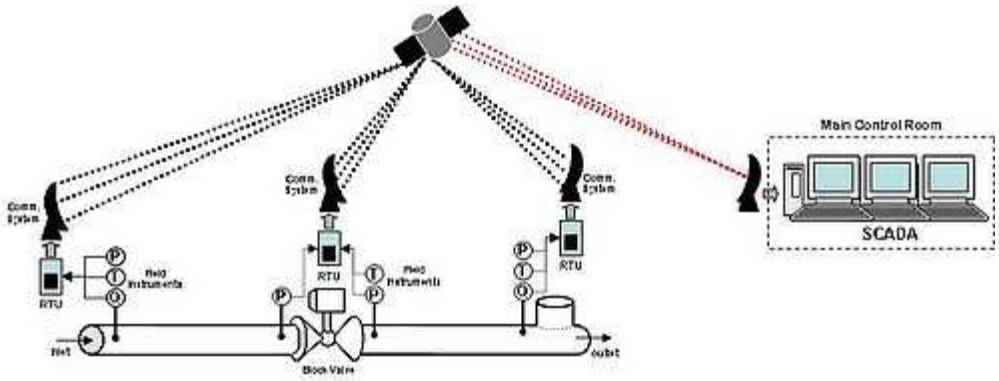

Figure 13: **A SCADA [Supervisory Control And Data Acquisition] Diagram**