Domain Analysis

An Analysis Process Model

Dines Bjørner

Fredscej 11, DK-2849 Holte, Denmark

May 30, 2013

Summary

- We present a summary, Sect. 2., of a rather complex structure of domain analysis &[1] description concepts: techniques and tools.
- And we link, in Sect. 3., these concepts, embodied in *domain analysis prompt*s and *domain description prompt*s, in a model of how a diligent **domain analyser cum describer** would use them.
- We claim that both sections, Sects. 2.–3. are contributions to a methodology of software engineering.
- This talk is based on that of [2013da].

[1] We use the empersand '&' between two terms a and b to emphasize that the term $a\&b$ is one.

1. Introduction

- Before **software** can be designed
  we must have a reasonably good grasp of its requirements.
- Before **requirements** can be prescribed
  we must have a reasonably good grasp of the **domain** in which the software is to reside.
- So we turn to **domain analysis & description** as a means to obtain and record that 'grasp'.
- In this talk we summarise an approach to **domain analysis & description** recorded in more detail in [2013da].
- Thus this talk is based on [2013da].

- This talk is one in a series of talks on **domain science & engineering**.
  - In `[dines:facs:2008]` we present techniques related to the analysis and description of **domain facets**.
  - In `[dines:ictac:2007]` we investigate some research issues of **domain science**.
  - And in `[dines:ugo65:2008]` we show how to systematically "transform" **domain descriptions** into **requirements prescriptions**.
  - The paper `[dines:humanity:2012]` examines possible contributions of domain science & engineering th *computation for the humanities*.

- It is expected that the present talk may be followed by respective ("spin-off") talks on
  - *Perdurants* [2013da-perd],
  - *A Formal Model of Prompts* [2013da-prompts],
  - *Domain Facets* (cf. [dines:facs:2008]) [2013da-facets], and
  - *On Deriving Requirements From Domain Descriptions* (cf. [dines:ugo65:2008]) [2013da-reqs].

- The structure of this paper is as follows:
  - First, Sect. 2 we present a terse summary of a system of **domain analysis & description** concepts focused on **endurants**.
    - This summary is rather terse,
    - and is a *"tour de force"*.
    - Section 2 is one of the two main sections of this talk.

- Section 3 suggest a formal-looking model of the structure of *domain analysis prompt*s and *domain description prompt*s introduced in Sect. 2.
  - It is not a formalisation of domains, but of the domain analysis & description process.
  - Domains are usually not computationally tractable.
  - Less so is the domain analysis & description processes.

- Finally, Sect. 4 concludes this talk.
- An appendix, Appendix A, presents a domain description of a [class of] *pipeline systems*.
  - Some seminars over the underlying paper may start by a brief presentation of this model.
  - The reader is invited to browse this *pipeline system* model before, during and/or after reading Sects. 2–3.

2. A Summary of The TripTych Domain Analysis Approach

2.1. Hierarchical versus Compositional Analysis & Description

- In this talk we choose, what we shall call, a **'hierarchical analysis'** approach which is
  - was based on decomposing an understanding of a domain
  - from the "overall domain" into its components,
  - and these, if not atomic, into their subcomponents.
- In contrast we could have chosen a **'compositional analysis'** approach
  - which starts with an understanding of a domain
  - from its atomic endurants
  - and composes these into composite ones,
  - finally ending up with an "overall domain" description.

2.2. Domains

- A **'domain'** is characterised by its
  - observable, i.e., manifest *entities*
  - and their *qualities*. [2]

Example 1. Domains:

- *a road net,*
- *a container line,*
- *a pipeline,*
- *a hospital* ■ [3]

[2] **Definitions** start with a single quoted **'term'** and conclude with a •

[3] **Examples** conclude with a ■

2.3. Sorts, Types and Domain Analysis

- By a **'sort'** (or **'type'**) we shall understand
  - the largest set of entities
  - all of which have the same qualities[4].

Example 2. Sorts:

- Links of any road net form a sort.
- So does hubs.
- The largest set of (well-formed) collections of links form a sort.
- So does similar collections of hubs.
- The largest set of road nets (containing well-formed collections of hubs and links) form a sort.

[4] Taking a sort (type) to be *the largest set of entities all of which have the same qualities* reflects Ganter & Wille's notion of a **'formal concept'** [GanterWille:ConceptualAnalysis1999].

- By **'domain analysis'** we shall understand
  - a process whereby a domain analyser
  - groups entities of a domain
  - into sorts (and types).
- The rest of this talk will outline a class of domain analysis
  - principles,
  - techniques and
  - tools.

2.4. Entities and Qualities

2.4.1. Entities

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

Example 3. Entities:

- *a road net, a link*[5] *of a road net, a hub*[6] *of a road net;* and
- *insertion of a link in a road net, disappearance of a link of a road net, and the movement of a vehicle on a road net*.

[5] A link: a street segment between two adjacent hubs
[6] A hub: an intersection of street segments

2.4.2. Qualities

- By a **'quality'** of an entity we shall understand
  - a **property** that can be
  - given a name and
  - precisely measured by physical instruments
  - or otherwise identified.

- **Example 4.** Quality Names:
  - *cadestral location of a hub*,
  - *hub state*[7],
  - *hub state space*[8],
  - etcetera ■
- **Example 5.** Quality Values:
  - *the name of a road net*,
  - *the ownership of a road net*,
  - *the length of a link*,
  - *the location of a hub*, etcetera ■

[7] From which links can one reach which links at a given time.
[8] Set of all hub states over time.

2.5. Endurants and Perdurants

- Entities are either **endurants** or are **perdurants**.

2.5.1. Endurants

- By an **'endurant entity'** (or just, an endurant) we shall understand
  - that can be observed or conceived,
  - as a "complete thing",
  - at no matter which given snapshot of time.

  Were we to "freeze" time
  - we would still be able to observe the entire endurant.
- Thus the method provides a *domain analysis prompt*:
  - `is_endurant` where
  - `is_endurant`(e) holds if entity e is an endurant.

Example 6. Endurants: Items (a–b–c) of Example on Slide 14 are endurants; so are the pipes, valves, and pumps of a pipeline.

2.5.2. Perdurants

- By a **'perdurant entity'** (or just, an perdurant) we shall understand
  - for which only a fragment exists
    if we look at or touch them
    at any given snapshot in time, that is,
  - where 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.

Example 7. Perdurants: Items (d–e–f) of Example on Slide 14 are perdurants; so are the insertion of a hub, removal of a link, etcetera ■

2.6. Discrete and Continuous Endurants

- Entities are either discrete or are continuous.

2.6.1. Discrete Endurants

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

  We use the term **'part'** for discrete endurants,

  that is: $\texttt{is\_part}(p) \equiv \texttt{is\_endurant}(p) \wedge \texttt{is\_discrete}(p)$.
- Thus the method provides a *domain analysis prompt*:
  - `is_discrete` where
  - `is_discrete`(e) holds if entity e is discrete.

Example 8. Discrete Endurants: The examples of Example on Slide 18 are all discrete endurants ■

2.6.2. Continuous Endurants

- 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 continuous.

Example 9. Continuous Endurants: The pipes, valves, pumps, etc., of Example on Slide 18 may contain oil; water of a hydro electric power plant is also a material (i.e., a continuous endurant) ■

2.7. Discrete and Continuous Perdurants

- We are not covering perdurants in this talk.

2.8. Atomic and Composite Discrete Endurants

- Discrete endurants are
  - either **atomic**
  - or **composite**.

2.8.1. Atomic Endurants

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

Example 10. Atomic Parts: Examples of atomic parts of the above mentioned domains are:

- aircraft (of air traffic),
- demand/deposit accounts (of banks),
- containers (of container lines),
- documents (of document systems),
- hubs, links and vehicles (of road traffic),
- patients, medical staff and beds (of hospitals),
- pipes, valves and pumps (of pipeline systems), and
- rail units and locomotives (of railway systems) ■

2.8.2. Composite Endurants

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

Example 11. Composite Parts: Examples of atomic parts of the above mentioned domains are:

- airports and air lanes (of air traffic),
- banks (of a financial service industry),
- container vessels (of container lines),
- dossiers of documents (of document systems),
- routes (of road nets),
- medical wards (of hospitals),
- pipelines (of pipeline systems), and
- trains, rail lines and train stations (of railway systems) ■

2.9. Part Observers

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

2.9.1. Composite Sorts

- For composite parts, p, the *domain description prompt*
  - `observe_parts`(p)
- yields some *formal description text* according to the followig *schema*:
- **type P_1, P_2, ..., P_n;**
  value obs_P_1: $\mathsf{P}\to\mathsf{P}_1$, obs_$\mathsf{P}_2$: $\mathsf{P}\to\mathsf{P}_2$,...,obs_$\mathsf{P}_n$: $\mathsf{P}\to\mathsf{P}_n$;
- where sorts P_1, P_2, ..., P_n must be disjoint.
  - A proof obligation
  - may need be discharged
  - to secure disjointness.

2.9.2. Sort Models

- A part sort is an abstract type.
  - Some part sorts, Ps, may have a concrete type model.
  - Here we consider only two such models:
    - one model is as sets of parts of sort Ps.
    - the other model has parts being of either of two or more alternative, disjoint sorts.
- 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 followig *schema*,
  - either
    - **type A, B, ..., C, T = $\mathcal{E}$(A,B,...,C);**
      value obs_T: P → T;
      where A,B,...,C are either (new) part sorts or are auxiliary (abstract or concrete) types[9];
  - or:
    - **type T = P1|P2|...|PN,**
      P_1,P_2,...,P_n,
      P1::mkP1(s_p:P_1),P2::mkP1(s_p:P_2),...,PN::mkP1(s_p:P_n);
      value obs_T: P→T;

[9] The *domain analysis prompt*: `sorts_of`(t) yields a subset of {A,B,...,C}.

2.10. 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 followig *schema*:
- **type** M_1, M_2, ..., M_m;
  value obs_M_1: P $\to$ M_1, **obs_**M_2: P $\to$ M_2, ..., **obs_**M_m: P $\to$ M_m;
- where values, m_i, of type M_i satisfy is_material(m) for all i;
- and where M_1, M_2, ..., M_m must be disjoint sorts.

Example 12. Part Materials:

- The pipeline parts p
  - pipes,
  - valves,
  - pumps,

  etc., contains some
  - either liquid material, say crude oil.
  - or gaseous material, say natural gas ■

- Some material m of sort M may contain parts.
  - The *domain analysis prompt*
    - has_parts(m)
  - holds if material m contains one or more parts.
- The *domain description prompt*
  - observe_part_sorts(m)
- yields some *formal description text* according to the followig *schema*:
- **type P_1, P_2, ..., P_n;**
  value obs_P_1: M→P_1, obs_P_2: M→P_2,...,obs_P_m: M→P_m;
- where values, p_i, of type P_i satisfy is_part(p_i) for all i;
- and where P_1, P_2, ..., P_n must be disjoint sorts.

Example 13. Material and Part Relations:

- A global transport system can, for example, be described as
  - primarily containing
    - navigable waters,
    - land areas and
    - air

    — as three major collections of parts.
  - Navigable waters contain a number of
    - "neighbouring" oceans,
    - channels,
    - canals,
    - rivers and
    - lakes reachable by canals or rivers from other navigable waters

    (all of which are parts).

- The part sorts of navigable waters has water materials.
- All water materials has (zero or more) parts such as vessels and sea-ports.
- Land areas contain
  - continents,
    some of which are neighbouring (parts),
  - while some are isolated
    (that is, being islands not "border–"connected to other continents).
- Some land areas contain harbour.
  - Harbours and seaports are overlapping parts sharing many attributes.
  - And harbours and seaports are connected to road and rail nets.
- Etcetera, etcetera.

• The above example, Example 13, help motivate the concept of **mereology** (see below).

2.11. Endurant Properties

2.11.1. External and Internal 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).

2.12. Unique Identifiers

- Without loss of generality we can assume that every part has a unique identifier[10].
  - 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.
- The *domain description prompt*:
  - `observe_unique_identifier(`p`)`
- yields some *formal description text* according to the followig *schema*:
- **type PI; value uid_P: P → PI;**

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

Example 14. Unique Identifiers:

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

2.13. Mereology

- By **'mereology'** [Lesniewski1] we shall understand
  - the study, knowledge and practice of
  - parts,
  - their relations to other parts
  - and "the whole".
- Part relations are such as:
  - two or more parts being connected,
  - one part being embedded within another part, and
  - two or more parts sharing (other) attributes.

Example 15. Mereology:

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

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

  can then be invoked and
- yields some *formal description text* according to the followig *schema*:
- **type MT = $\mathcal{E}$(PI$_A$,PI$_B$,...,PI$_C$); value mereo_P: P → MT;**
  where $\mathcal{E}(...)$ is some type expression over unique identifier types of one or more part sorts.

- Mereologies are expressed in terms of structures of unique part identifiers.
- Usually mereologies are constrained. Constraints express
  - that a mereology's unique part identifiers must indeed reference existing parts, but also
  - that these mereology identifiers "define" a proper structuring of parts.

Example 16. Mereology Constraints:

- We continue our line of examples of road net endurants, but now a bit more systematically:
  - A road net, n:N, contains a pair, (HS,LS), of sets Hs of hubs h:H and sets Ls of links.
  - The mereology of links must identify exactly two hubs of the road net,
  - the mereology of hubs must identify links of the road net,
  - so connected hubs and links must have commensurate mereologies ■

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

2.14. Attributes

- Attributes are what really endows parts with qualities.
  - The external properties[11]
  - 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 attributes.

11

- `is_discrete`,
- `is_continuous`,
- `is_atomic`,
- `is_composite` and
- `has_material`.

- By a **'part attribute'**, or just an **'attribute'**, we shall understand
  - a property that is associated with a part p of sort P,
    - and if removed from part p,
    - that part would no longer be part p
    - but may be a part of some other sort P'; and
  - where that property itself has no physical extent (i.e., volume),
    - as the part may have,
    - but may be measurable by physical means.

Example 17. **Attributes:** Some attributes

- of road net hubs are
  - ⟐ location,
  - ⟐ hub state[12],
  - ⟐ hub state space[13],

  and
- of road net links are
  - ⟐ location,
  - ⟐ length,
  - ⟐ link state[14],
  - ⟐ link state space[15],

  etcetera ■.

[12] Hub state: a set of pairs of unique identifiers of actually connected links.
[13] Hub state space: a set of hub states that a hub states may range over.
[14] Link state: a set of pairs of unique identifiers of actually connected hubs.
[15] Link state space: a set of link states that a link state may range over.

- The *domain description prompt*
  - observe_attributes(p)
- yields some *formal description text* according to the followig *schema*:
  - **type A$_1$, A$_2$, ..., A$_n$, ATTR;**
    value attr_A$_1$:P→A$_1$, attr_A$_2$:P→A$_2$, ..., attr_A$_n$:P→A$_n$,
    attr_ATTR:P→ATTR;
  - where **for** ∀ p:P, **attr**_A$_i$(**attr**_ATTR(p)) ≡ **attr**_A$_i$(p).

2.14.1. Shared Attributes

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

2.15. Discussion

- We have left out any coverage of perdurant entities.
  - For the time we refer to Sect. 5 of [2013da],
  - hoping to further develop that section's understanding
  - into a forthcoming study [2013da-perd].

3. A Prompt & Description 'Method'

3.1. A Summary of Prompts

- In the previous section we outlined two classes of prompts:
  - ⋈ the *domain analysis prompt*s:

attribute_ names [XII], 16
has_ concrete_ type [XI], 10
has_ materials [XIV], 25
has_ mereology [XIII], 22
is_ atomic [VIII], 7
is_ composite [IX], 7
is_ continuous [V], 6
is_ discrete [IV], 6
is_ endurant [II], 6
is_ entity [I], 5
is_ material [VII], 7
is_ part [VI], 7
is_ perdurant [III], 6
observe_ parts [X], 8

  - ⋈ and the *domain description prompt*s:

observe_ attributes [4], 16
observe_ material_ sorts [6], 25
observe_ mereology [5], 22
observe_ part_ sorts [1], 8
observe_ part_ type [2], 10
observe_ unique_ identifier [3], 14

- These prompts are imposed upon the domain analyser cum describer.
- They are "figuratively" applied to the domain.
- Their orderly, sequenced application
  - follows the method hinted at in the previous section and
  - expressed in a pseudo-formal notation in this section.
- The notation looks formal
  - but since we have not formalised these prompts
  - it is only pseudo-formal.
- In `[2013da-prompts]` we shall formalise these prompts.

3.2. Preliminaries

- Let P be a sort, that is, a collection of endurants.
- By ηP we shall understand a syntactic quantity: the name of P.
- By ιp:P we shall understand the semantic quantity: an (arbitrarily selected) endurant in P.
- And by $\eta^{-1}\eta$P we shall understand P.

- To guide our analysis & description process we decompose it into steps.
  - Each step "handles" a sort P:P or a material m:M.
  - Steps handling discovery of composite sorts generates a set of sort names $\eta\mathsf{P}_1$, $\eta\mathsf{P}_2$, ..., $\eta\mathsf{P}_n$ and $\eta\mathsf{M}_1$, $\eta\mathsf{M}_2$, ..., $\eta\mathsf{M}_n$.
  - These are put in a reservoir for sorts to be inspected.
  - The handled sort $\eta\mathsf{P}$ or $\eta\mathsf{M}$ is removed from that reservoir.
  - Handling of material sorts concerns only their attributes.
  - Each domain description prompt results in domain specification text (here we show only the formal texts) being deposited in the domain description reservoir, a global variable τ.

- The clause:
  - domain_description_prompt(p) : τ := $\tau \oplus$ ["text ; "]
- means that
  - the formal text "text ; " is joined to the global variable τ
  - where that "text ; " is prompted by domain_description_prompt(p).
- The meaning of $\oplus$ will be discussed at the end of this section.

3.3. Initialising the Domain Analysis & Description Process

- We remind the audience that we are dealing only with endurant domain entities.
- The domain analysis approach covered in Sect. 2
  - was based on decomposing an understanding of a domain
  - from the "overall domain" into its components,
  - and these, if not atomic, into their subcomponents.
- So we need to initialise the domain analysis & description by selecting (or choosing) the domain Δ.

- Here is how we think of that "initialisation" process.
  - The domain analyser[s] & describer[s] spends some time focusing on the domain,
  - maybe at the "white board",
  - rambling, perhaps in an un-structured manner,
  - across its domain, Δ, and its sub domains.
  - Informally jotting down more-or-less final sort names,
  - building, in the domain analysers' & describers' mind
  - an image of that domain.

- After some time doing this
  - the domain analyser[s] & describer[s] is/are ready.
  - An image of the domain
  - in the form of "a domain" endurant, δ:Δ.
  - Those are the quantities,
    - $\eta\Delta$ (name of Δ) [Item 1 on the following slide] and
    - ιp:P (for (δ:Δ)) [Item 8 on Slide 62],

    referred to below.
- Thus this initialisation process is truly a creative one.

3.4. A Domain Analysis & Description State

1. A global variable α**ps** will accumulate all the sort names being discovered.
2. A global variable ν**ps** will hold names of sorts yet to be analysed and described.
3. A global variable τ will hold the (so far) generated (in this case only) formal domain description text.

variable

1. αps := [$\eta\Delta$] ηP-**set** or ηP*
2. νps := [$\eta\Delta$] (ηP|ηM)-**set** or (ηP|ηM)*
3. τ := [ ] **Text-set** or **Text***

- We shall explain the use of [...]s and the operations of \ and $\oplus$ on the above variables in Sect. 4.10.5 Slide 85.

3.5. Analysis & Description of Endurants

4. To analyse and describe endurants means to first
5. examine those endurant which have yet to be so analysed and described
6. by selecting (and removing from νps) a yet unexamined sort (by name);
7. then analyse and describe an endurant entity (ιp:P) of that sort — this analysis, when applied to composite parts, leads to the insertion of zero[16] or more sort names[17];
8. then to analyse and describe the mereology of each part sort,
9. and finally to analyse and describe the attributes of each sort.

[16] If the sub-parts of p are all either atomic or already analysed, then no new sort names are added to the repository νps).

[17] These new sort names are then "picked-up" for sort analysis &c. in a next iteration of the while loop.

```
value
4.  analyse_and_describe_endurants: Unit → Unit
4.  analyse_and_describe_endurants() ≡
5.    while ∼is_empty(νps) do
6.      let ηS = select_and_remove_ηS() in
7.      analyse_and_describe_endurant_sort(ιs:S) end end ;
8.    for all ηP · ηP ∈ αps do analyse_and_describe_mereology(ιp:P) end
9.    for all ηP · ηP ∈ αps do analyse_and_describe_attributes(ιp:P) end
```

- The ι of Items 7, 8 and 9 are crucial.
  - The domain analyser is focused on sort S (and P)
  - and is "directed" (by those items)
  - to choose (select) an endurant ιs (ιp) of that sort.
  - The ability of the domain analyser to find such an entity
  - is a measure of that person's professional creativity.

- As was indicated in Sect. 2.,
  - the mereology of a part may involve unique identifiers of any part sort, hence must be done after all such part sort unique identifiers have been identified.
  - Similarly for attributes which also may involve unique identifiers
  - Each iteration of analyse_and_describe_endurant_sort(ιp:P)
    - involves the selection of a sort (by name)
    - (which is that of either a part sort or a material sort)
    - with this sort name then being removed.

10. The selection occurs from the global state (hence: ()) and changes that (hence **Unit**).
11. The affected global state component is that of the reservoir, νps.

value

10. select_and_remove_ηS: **Unit** $\rightarrow$ ηP
10. select_and_remove_ηS() $\equiv$
11. **let** ηS $\cdot$ ηS $\in$ νps **in** νps := νps $\setminus$ {ηS} ; ηS **end**

- The analysis and description of all sorts
  - also performs an analysis and description of their
  - possible unique identifiers (if part sorts) and
  - attributes.
- The analysis and description of sort mereologies potentially requires the unique identifiers of any set of sorts. Therefore the analysis and description of sort mereologies follows that of analysis and description of all sorts.

12. To analyse and describe an endurant
13. is to find out whether it is a part.
14. If so then it is to analyse and describe it as a part,
15. else it is to analyse and describe it as a material.

```
12.  analyse_and_describe_endurant_sort: (P|M) → Unit
12.  analyse_and_describe_endurant_sort(e:(P|M)) ≡
13.    if is_part(e)
13.      assert: is_part(e) ≡ is_endurant(e)∧is_discrete(e)
14.      then analyse_and_describe_part_sort(e:P)
15.      else analyse_and_describe_material_parts(e:M)
12.    end
```

3.5.1. Analysis & Description of Part Sorts

16. The analysis and description of a part sort
17. is based on there being a set, ps, of parts to analyse —
18. of which an archetypal one, p′, is arbitrarily selected.
19. analyse and describe part p′

```
16.  analyse_and_describe_part_sort: P → Unit
16.  analyse_and_describe_part_sort(p:P) ≡
17.    let ps = observe_parts(p) in
18.    let p′:P · p′ ∈ ps in
19.    analyse_and_describe_part(p′)
16.    end end
```

20. The analysis (&c.) of a part
21. first analyses and describes its unique identifiers.
22. If atomic
23. and
24. if the part embodies materials,
25. we analyse and describe these.
26. If not atomic then the part is composite
27. and is analysed and described as such.

```
20.  analyse_and_describe_part: P → Unit
20.  analyse_and_describe_part(p) ≡
21.    analyse_and_describe_unique_identifier(p) ;
22.    if is_atomic(p)
23.      then
24.        if has_materials(p)
25.          then analyse_and_describe_part_materials(p) end
26.      else assert: is_composite(p)
27.        analyse_and_describe_composite_endurant(p) end
20.    pre: is_discrete(p)
```

- We do not associate materials with composite parts.

3.5.2. Analysis & Description of Part Materials

28. The analysis and description of the material part sorts, one or more, of atomic parts p of sort P containing such materials,
29. simply observes the material sorts of p,
30. that is generates the one or more continuous endurants
31. and the corresponding observer function text.
32. The reservoir of sorts to be inspected is augmented by the material sorts — except if already previously entered (the $\setminus$ α**ps** clause).

```
28.  analyse_and_describe_part_materials: P → Unit
28.  analyse_and_describe_part_materials(p) ≡
29.      observe_material_sorts(p) :
30.      τ := τ ⊕ [”type M1,M2,...,Mm;
31.                value obs_M1:P→M1,obs_M2:P→M2,...,obs_Mm:P→Mm;” ]
32.      νps := νps ⊕ ([ M1,M2,...,Mm ] \ αps)
28.      pre: has_materials(p)
```

3.5.3. Analysis & Description of Material Parts

33. To analyse and describe materials, m, i.e., continuous endurants,
34. is only necessary if m has parts.
35. Then we observe the sorts of these parts.
36. The identified part sort names update both name reservoirs.

```
33.  analyse_and_describe_material_parts: M → Unit
33.  analyse_and_describe_material_parts(m:M) ≡
34.    if has_parts(m)
35.      then observe_part_sorts(m):
35.             τ := τ ⊕ [" type P1,P2,...,PN ;
35.                      value obs_Pi: M→Pi i:{1..N};" ]
36.        ‖ νps := νps ⊕ ([ ηP1,ηP2,...,ηPN ]\ αps)
36.        ‖ αps := αps ⊕ [ ηP1,ηP2,...,ηPN ]
33.      end
33.    assert: is_continuous(m)
```

3.5.4. Analysis & Description of Composite Endurants

37. To analyse and describe a composite endurant of sort P
38. is to (we choose first) to analyse and describe the unique identifier of that composite endurant,
39. then to analyse and describe the sort. If the sort has a concrete type
40. then we analyse and describe that concrete sort type
41. else we analyse and describe the abstract sort.

```
37.  analyse_and_describe_composite_endurant: P → Unit
37.  analyse_and_describe_composite_endurant(p) ≡
38.    analyse_and_describe_unique_identifier(p) ;
39.    if has_concrete_type(p)
40.      then analyse_and_describe_concrete_sort(p)
41.      else analyse_and_describe_abstract_sort(p)
39.    end
```

3.5.5. Analysis & Description of Concrete Sort Types

42. The concrete sort type being analysed and described is
43. either
44. expressible by some compound type expression
43. or is
45. expressible by some alternative type expression.

```
42.  analyse_and_describe_concrete_sort: P → Unit
42.  analyse_and_describe_concrete_sort(p:P) ≡
44.    analyse_and_describe_concrete_compound_type(p)
43.    ⌈⌉
45.    analyse_and_describe_concrete_alternative_type(p)
42.    pre: has_concrete_type(p)
```

46. The concrete compound sort type
47. is expressible by some simple type expression, T=$\mathcal{E}$(Q,R,...,S) over either concrete types or existing or new sorts Q, R, ..., S.
48. The emerging sort types are identified
49. and assigned to both νps
50. and αps.

44. analyse_and_describe_concrete_compound_type: P → **Unit**
44. analyse_and_describe_concrete_compound_type(p:P) ≡
46. **observe_part_type**(p):
46. τ := τ ⊕ [”**type** Q,R,..,S, T = $\mathcal{E}$(Q,R,...,S);
46. **value obs**_T: P → T ;”] ;
47. **let** {P_a,P_b,...,P_c} = sorts_of({Q,R,...,S})
48. **assert:** {P_a,P_b,...,P_c} ⊆ {Q,R,...,S} **in**
49. ν**ps** := ν**ps** ⊕ [ηP_a, ηP_b, ..., ηP_c] ‖
50. α**ps** := α**ps** ⊕ ([ηP_a, ηP_b, ..., ηP_c] \ α**ps**) **end**
44. **pre**: **has_concrete_type**(p)

51. The concrete alternative sort type expression
52. is expressible by an alternative type expression T=P1|P2|...|PN where each of the alternative types is made disjoint wrt. existing types by means of the description language Pi::mkPi(s_u:P_i) construction.
53. The emerging sort types are identified and assigned
54. to both νps
55. and αps.

45. analyse_and_describe_concrete_alternative_type: P → **Unit**
45. analyse_and_describe_concrete_alternative_type(p:P) ≡
51. **observe_part_type**(p):
52. τ := τ ⊕ ["**type** T=P1 | P2 | ... | PN, Pi::mkPi(s_u:P_i) (1≤i≤N);
52. **value obs**_T: P→T ;"] ;
53. **let** {P_a,P_b,...,P_c} = sorts_of({P_i|1≤i≤n})
53. **assert:** {P_a,P_b,...,P_c} ⊆ {P_i|1≤i≤n} **in**
54. ν**ps** := ν**ps** ⊕ ([ηP_a, ηP_b, ..., ηP_c] \ α**ps**) ‖
55. α**ps** := α**ps** ⊕ [ηP_a, ηP_b, ..., ηP_c] **end**
42. **pre**: **has_concrete_type**(p)

3.5.6. Analysis & Description of Abstract Sorts

56. To analyse and describe an abstract sort
57. amounts to observe part sorts and to
58. update the sort name repositories.

```
56.  analyse_and_describe_abstract_sort: P → Unit
56.  analyse_and_describe_abstract_sort(p:P) ≡
57.    observe_part_sorts(p):
57.      τ := τ ⊕ [”type P1, P2, ..., Pn;
57.                value obs_Pi:P→Pi (0≤i≤n);” ]
58.    ‖ νps := νps ⊕ ([ηP1, ηP2, ..., ηPn] \ αps)
58.    ‖ αps := αps ⊕ [ηP1, ηP2, ..., ηPn]
```

3.5.7. Analysis & Description of Unique Identifiers

59. To analyse and describe the unique identifier of parts of sort P is
60. to observe the unique identifier of parts of sort P
61. where we assume that all parts have unique identifiers.

```
59.  analyse_and_describe_unique_identifier: P → Unit
59.  analyse_and_describe_unique_identifier(p) ≡
60.    observe_unique_identifier(p):
60.      τ := τ ⊕ [ "type PI; value uid_P:P→PI;" ]
61.    assert: has_unique_identifier(p)
```

3.5.8. Analysis & Description of Mereologies

62. To analyse and describe a part mereology
63. if it has one
64. amounts to observe that mereology
65. and otherwise do nothing.
66. The analysed quantity must be a part.

```
62. analyse_and_describe_mereology: P → Unit
62. analyse_and_describe_mereology(p) ≡
63.     if has_mereology(p)
64.         then observe_mereology(p) :
64.             τ := τ ⊕ "type MT = ℰ(PI_a,PI_b,...,PI_c) ;
64.                      value mereo_P: P→MT ;"
65.         else skip end
62.     pre: is_part(p)
```

3.5.9. Analysis & Description of Part Attributes

67. To analyse and describe the attributes of parts of sort P is
68. to observe the attributes of parts of sort P
69. where we assume that all parts have attributes.

```
67.  analyse_and_describe_part_attributes: P → Unit
67.  analyse_and_describe_part_attributes(p) ≡
68.    observe_attributes(p):
68.      τ := τ ⊕ [ "type A1, A2,..., Am ;
68.                 value attr_A1:P→A1,attr_A2:P→A2,...,attr_Am:P→Am;" ]
69.    assert: has_attributes(p)
```

3.6. Discussion of The Model

- The above model
  - lacks a formal understanding of the individual prompts as listed in Sect. 3.1;
  - such an understanding is attempted in [2013da-prompts].

3.6.1. Termination

- The sort name reservoir ν**ps**
  - is "reduced" by one name in each iteration of the **while** loop of the analyse_and_describe_endurants, cf. Item 6 on Slide 62,
  - and is augmented by new part and material sort names in some iterations of that loop, cf. formula Items
    - 32 on Slide 70,
    - 36 on Slide 71,
    - 49 on Slide 74,
    - 54 on Slide 76 and
    - 49 on Slide 74.
  - It remains to prove that the analysis & description process terminates.
-

3.6.2. Axioms and Proof Obligations

- We have omitted from the above (and also in Sect.) treatment of
  - axioms concerning well-formedness of parts, materials and attributes and
  - proof obligations concerning disjointness of observed part and material sorts and attribute types.
- A more proper treatment would entail adding a line
  - of proof obligation text right after Item lines
    - 65 on Slide 80 and
    - 68 on Slide 81.
  - and of axiom text right after Item lines
    - 31 (Slide 70),
    - 35 (Slide 71),
    - 46 (Slide 74),
    - 48 (Slide 74),
    - 60 (Slide 79) and
    - 68 (Slide 81).
  - No axiom is needed in connection with Item line 52 on Slide 76.
- [2013da] covers axioms and proof obligations in some detail.

3.6.3. Order of Analysis & Description: A Meaning of '⊕'

- The variables αps, νps and τ are defined to hold either sets or lists.
- The operator $\oplus$ can be thought of
  - as either set union ($\cup$ and [,]≡{, }) — in which case the domain description text in τ is a set of domain description texts
  - or as list concatenation ($\frown$ and [,]≡⟨,⟩) of domain description texts.
  - The operator $\ell_1 \oplus \ell_2$ now has at least two interpretations:
    - either $\ell_1 \frown \ell_2$
    - or $\ell_2 \frown \ell_1$.
  - In the case of lists the $\oplus$ (i.e., $\frown$) does not (suffix or prefix) append ℓ_2 elements already in ℓ_1.

- The select_and_remove_ηP function on Slide 65 applies to the set interpretation.
- A list interpretation is:

value
6. select_and_remove_ηP: **Unit** $\to$ ηP
6. select_and_remove_ηP() $\equiv$
6. **let** ηP = **hd** νps **in** νps := **tl** νps; ηP **end**

- In the first case ($\ell_1 \widehat{} \ell_2$) the analysis and description process proceeds from the root, breadth first,
- In the second case ($\ell_2 \widehat{} \ell_1$) the analysis and description process proceeds from the root, depth first.

.

3.6.4. Laws of Description Prompts

- The domain 'method' outlined in the previous section suggests that many different orders of analysis & description may be possible.
- But are they? That is, will they all result in "similar" descriptions?
- That is, if $\mathcal{D}_a$ and $\mathcal{D}_b$
  - are two domain description prompts
  - where $\mathcal{D}_a$ and $\mathcal{D}_b$
  - can be pursued in any order
  - will that yield the same description?
  - And what do we mean by
    - 'can be pursued in any order', and
    - 'same description'?

- Let us assume that sort P decomposes into sorts P_a and P_b (etcetera).
  - Let us assume that the
    - domain description prompt $\mathcal{D}_a$ is related to the description of P_a and
    - $\mathcal{D}_b$ to P_b.
  - Here we would expect $\mathcal{D}_a$ and $\mathcal{D}_b$ to commute, that is
    - $\mathcal{D}_a;\mathcal{D}_b$ yields same result
    - as does $\mathcal{D}_b;\mathcal{D}_a$.
  - In `[Kiev:2010ptII]` we made an early exploration of such laws of domain description prompts.
- To answer these questions we need a reasonably precise model of domain prompts.
- We attempt such a model in `[2013da-prompts]`.

4. Conclusion

- It is time to conclude.

4.1. Comparison to Other Work

4.1.1. Domain Analysis

- In [2013da] we give, in its Sect. 6.3, a 3+ page comparison to the broader concept of 'domain analysis' as covered in almost 30 literature references.
- We claim, on this background, that our concept of domain analysis, as treated in Sect. 2, is sufficiently different (i.e., novel) as to warrant your attention!

4.1.2. Methodology

- We are not aware of any publications (other other than [Kiev:2010ptII]) that attempt for "formalise" core concepts of the notion of 'method'.

4.2. What Have We Achieved

4.2.1. Domain Analysis

- In Sect. 2 we have
  - presented a terse, five+ page, summary of a novel approach to domain analysis.
  - That this approach is different from other 'domain analysis' approaches is argued in [] Sect. 6.2]2013da.
  - The new aspects are:
    - the distinction between parts and materials,
    - the distinction between external and internal properties (Sect. 2.11.1),
    - the introduction of the concept of mereologies and
    - the therefrom separate treatment of attributes.
    - It seems to us that "conventional" domain analysis treated all endurant qualities as attributes.

- The many concepts,
  - endurants and perdurants,
  - discrete and continuous,
  - hence parts and materials,
  - atomic and composite,
  - uniqueness of parts,
  - mereology, and
  - shared attributes,

  we claim,
- are forced upon the analysis
  - by the nature of domains:
  - existing in some not necessarily computable reality.
- In this way the proposed domain analysis & description approach is new.

4.2.2. Methodology

- By a **'method'** we shall understand
  - a set of **principles** for
  - **selecting** and **applying**
  - **techniques** and **tools**
  - in order to **analyse** and **construct**
  - an **artifact**.

- Clearly Sect. 3 presents a partially instantiated framework for a formal model of a 'method':
  - Some principles are
    - **abstraction** (sorts in preference for concrete types),
    - **separation of concerns** (tackling endurants before perdurants),
    - commensurate narratives and formalisations,
    - tackling domain analysis
      - either "top-down", hierarchically from composite endurants,
      - or "bottom-up", compositionally, from atomic endurants,
      - or in some orderly combination of these;
    - Etcetera.

- Some techniques are
  - expressing axioms concerning well-formedness of mereologies and attribute values;
  - stating (and discharging) proof obligations securing disjointness of sorts;
  - etcetera.

- And some tools are
  - the *domain analysis prompt*s,
  - the *domain description prompt*s and
  - the description language (here RSL [RSL]).

- We claim that we have sketched a formalisation of a method for domain analysis and description.

- What is really new here is, as for domain analysis,
  - that the analysis & description process is applied to a domain,
  - that is, to our image of that domain,
  - something not necessarily computable,
  - and that our description therefore
    must not reduce the described domain to a computable artefact.

4.3. Future Work

- There remains
  - to conclude studies of,
  - to document and
  - publish

  treatments of the following related topics:
  - domain analysis of perdurants (actions, events and behaviours [[]Sect. 5]2013da) — including related *domain analysis prompt*s and *domain description prompt*s[18],
  - model(s) of prompts[19],
  - domain facets, cf. [dines:facs:2008][20], and
  - derivation of requirements from domain descriptions, cf. [dines:ugo65:2008][21]. .

[18] See forthcoming [2013da-perd]
[19] See forthcoming [2013da-prompts]
[20] See forthcoming [2013da-facets]
[21] See forthcoming [2013da-reqs]

4.4. Acknowledgements

5. Pipeline Endurants

- 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. 2,
    - 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 adviced in proper, industry-scale analyses & descriptions.

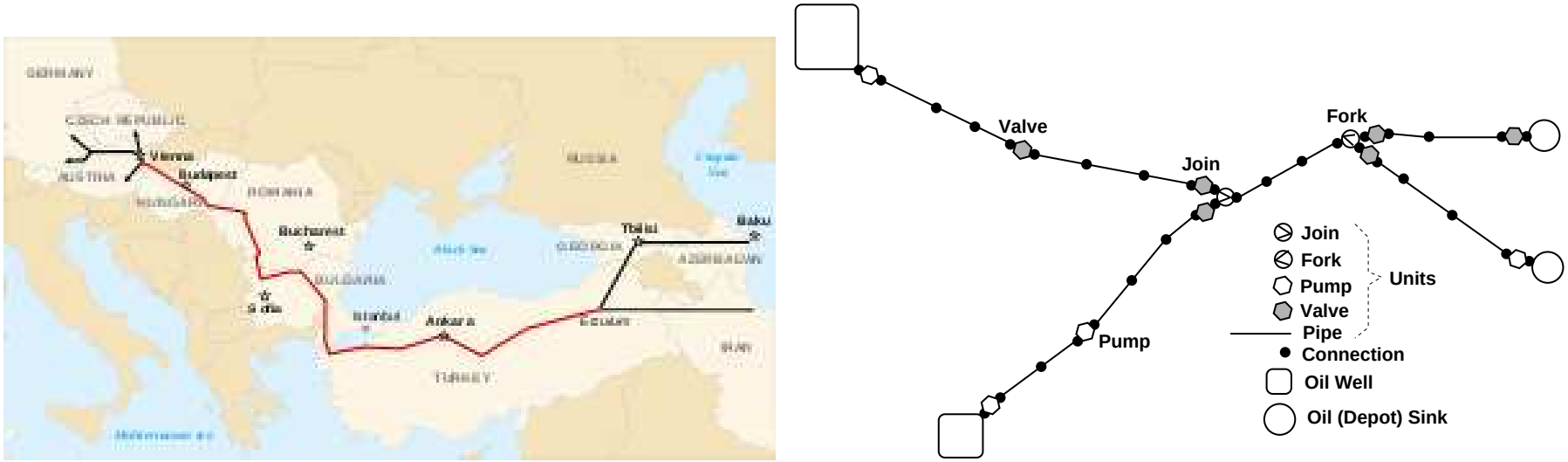

Figure 1: Oil or gas pipelines

- The description only covers a few aspects of endurants.

Figure 2: Some oil pipeline system units: pump, pipe, valve

5.1. Parts

70. A pipeline system contains a set of pipeline units and a pipeline system monitor.
71. The well-formedness of a pipeline system depends on its mereology and the routing of its pipes.
72. A pipeline unit is either a well, a pipe, a pump, a valve, a fork, a join, or a sink unit.
73. 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.

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

value
71. wf_PLS: PLS → **Bool**
71. wf_PLS(pls) ≡ wf_Mereology(pls) ∧ wf_Routes(pls)
70. obs_Us: PLS → U**-set**
70. obs_M: PLS → M

type
72. U = We | Pi | Pu | Va | Fo | Jo | Si
73. We :: Well
73. Pi :: Pipe
73. Va :: Valv
73. Fo :: Fork
73. Jo :: Join
73. Si :: Sink

5.2. Part Identification and Mereology

5.2.1. Unique Identification

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

type

74. UI

value

74. uid_UI: U → UI

axiom

74. ∀ pls:PLS,u,u′:U·{u,u′}⊆obs_Us(pls)⇒u≠u′⇒uid_UI(u)≠uid_UI(u′)

5.2.2. Unique Identifiers

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

value

75. xtr_UIs: PLS → UI-**set**
75. xtr_UIs(pls) ≡ {uid_UI(u)|u:U·u ∈ obs_Us(pls)}

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

theorem:

76. ∀ pls:PLS·**card** obs_Us(pl)=**card** xtr_UIs(pls)

5.2.3. Mereology

77. 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 sink is connected to exactly one input unit (and, hence, has no "output").

```
type
77.  MER = UI-set × UI-set
value
77.  mereo_U: U → MER
axiom
77.  wf_Mereology: PLS → Bool
77.  wf_Mereology(pls) ≡
77.    ∀ u:U•u ∈ obs_Us(pls)⇒
77.      let (iuis,ouis) = mereo_U(u) in iuis ∪ ouis ⊆ xtr_UIs(pls) ∧
77.        case (u,(card uius,card ouis)) of
77a..          (mk_We(we),(0,1)) → true,
77b..          (mk_Pi(pi),(1,1)) → true,
77c..          (mk_Pu(pu),(1,1)) → true,
77d..          (mk_Va(va),(1,1)) → true,
77e..          (mk_Fo(fo),(1,1)) → true,
77f..          (mk_Jo(jo),(1,1)) → true,
77g..          (mk_Si(si),(1,1)) → true,
77.          _ → false end end
```

5.3. Part Concepts

- An aspect of domain analysis & description that was not covered in Sect. 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.

5.3.1. Pipe Routes

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

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

type

78. R′ = U^ω
78. R = {| r:Route′·wf_Route(r) |}
79. RD = UI^ω

axiom

79. ∀ rd:RD · ∃ r:R·rd=descriptor(r)

value

79. descriptor: R → RD
79. descriptor(r) ≡ ⟨uid_UI(r[i])|i:**Nat**·1≤i≤**len** r⟩

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

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

81. 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 81a.–81c..

```
value
81.    Routes: PLS → RD-infset
81.    Routes(pls) ≡
81a.       let rs = ⟨⟩ ∪
81b.           {⟨uid_UI(u),uid_UI(u′)⟩|u,u′:U•{u,u′}⊆obs_Us(pls) ∧ adjacent(u,u′)}
81c.         ∪ {r^tl r′|r,r′:R•{r,r′}⊆rs}
81d.       in rs end
```

5.3.2. Well-formed Routes

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

value

82. acyclic_Route: R → **Bool**
82. acyclic_Route(r) ≡ ∼∃ i,j:**Nat**·{i,j}⊆**inds** r ∧ i≠j ∧ r[i]=r[j]

83. 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
83. wf_Routes: PLS → Bool
83. wf_Routes(pls) ≡
83.     non_circular(pls) ∧ are_embedded_in_well_to_sink_Routes(pls)

83. non_circular_PLS: PLS → Bool
83. non_circular_PLS(pls) ≡
83.     ∀ r:R·r ∈ routes(p)∧acyclic_Route(r)
```

84. 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
84.  well_to_sink_Routes: PLS → R-set
84.  well_to_sink_Routes(pls) ≡
84.    let rs = Routes(pls) in
84.    {r|r:R·r ∈ rs ∧ is_We(r[1]) ∧ is_Si(r[len r])} end
```

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

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

5.3.3. Embedded Routes

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

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

5.3.4. A Theorem

87. 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:

87. ∀ pls:PLS ·
87. **let** rs = Routes(pls),
87. wsrs = well_to_sink_Routes(pls) **in**
87a.. rs =
87b.. wsrs ∪
87c.. ∪ {{r′|r′:R · r′ ∈ embedded_Routes(r″)} | r″:R · r″ ∈ wsrs}
86. **end**

5.4. Materials

88. The only material of concern to pipelines is the gas[22] or liquid[23] which the pipes transport[24].

type

88. GoL

value

88. obs_GoL: U → GoL

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

5.5. Attributes

5.5.1. Part Attributes

89. 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
    e. flow (e.g., volume/second).

type

89a.. WellCap
89b.. LEN
89c.. Height
89d.. ValSta == open | close
89e.. Flow

90. Flows can be added (also distributively) and subtracted, and
91. flows can be compared.

value

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

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

value

92a.. attr_WellCap: We → WellCap
92b.. attr_LEN: Pi → LEN
92c.. attr_Height: Pu → Height
92d.. attr_ValSta: Va → VaSta
92e.. attr_In_Flow$_{\mathcal{L}}$: U → UI → Flow
92f.. attr_In_Leak$_{\mathcal{L}}$: U → UI → Flow
92g.. attr_Max_In_Leak$_{\mathcal{L}}$: U → UI → Flow
92h.. attr_body_Flow$_{\mathcal{L}}$: U → Flow
92i.. attr_body_Leak$_{\mathcal{L}}$: U → Flow
92j.. attr_Max_Flow$_{\mathcal{L}}$: U → Flow
92k.. attr_Out_Flow$_{\mathcal{L}}$: U → UI → Flow
92l.. attr_Out_Leak$_{\mathcal{L}}$: U → UI → Flow
92m.. attr_Max_Out_Leak$_{\mathcal{L}}$: U → UI → Flow

5.5.2. Flow Laws, I

93. "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:

93. ∀ u:U\We\Si ·
93. sum_in_leaks(u) ⊕ sum_in_flows(u) =
93. attr_body_Leak$_{\mathcal{L}}$(u) ⊕
93. 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_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_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_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_ℒ(u)(ui)|ui:UI•ui ∈ ouis} end
```

94. "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:

94. ∀ u,u′:U·adjacent(u,u′) ⇒
94. **let** (,ouis)=mereo_U(u), (iuis′,)=mereo_U(u′) **in**
94. **assert:** uid_U(u′) ∈ ouis ∧ uid_U(u) ∈ iuis′
94. attr_Out_Flow$_{\mathcal{L}}$(u)(uid_U(u′)) =
94. attr_In_Leak$_{\mathcal{L}}$(u)(uid_U(u))⊕attr_In_Flow$_{\mathcal{L}}$(u′)(uid_U(u)) **end**