The Manifest Domain Analysis & Description Approach to Implicit and Explicit Semantics

Invited Talk for the IMPEX Workshop, Xi’an, China, 16 November 2017

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

November 16, 2017: 00:23 am
Summary

- The domain analysis & description calculi introduced in [1] is shown to alleviate the issue of implicit semantics [2].
- The claim is made that domain descriptions, whether informal, or as also here, formal,
  amount to an explicit semantics for what is otherwise implicit if not described!
• I claim that [1] provides an answer to the claim in both [2, 3] that
  “The contexts of the systems in these cases are treated as second-class citizens . . . ”,
  respectively
  “In general, modeling languages are not equipped with resources, concepts or entities handling explicitly domain engineering features and characteristics (domain knowledge) in which the modeled systems evolve”.
Caveat!

- When I prepared this talk I was unaware of [3, Yamine Ait-Ameur and Dominique Méry, *Making explicit domain knowledge in formal system development*, Science of Computer Programming, 121, 120–127].

- I was first made aware of and given this paper Nov. 14, 2017.

- I apologize.
## Contents

**Introduction**  
- On the Issues of Implicit and Explicit Semantics  
- A Triptych of Software Engineering  
- Semantics  
- Method & Methodology  
- Computer & Computing Sciences  

### The Analysis & Description Prompts  

**Endurants: Parts, Components and Materials**  
- Internal Qualities  
  - Unique Identifiers  
  - Mereology  
  - Attributes  
- Attribute Categories  

**Description Axioms and Proof Obligations**  

**From Manifest Parts (Endurants) to Domain Behaviours (Perdurants)**  
- The Idea — by means of an example  
- Channels and Communication  
- Behaviour Signatures  
- Translation of Part Qualities  
- Part Behaviour Signatures  
- Behaviour Compilations  
- Atomic Behaviour Definitions  

**A Proof Obligation**  

**Calculations in Classical Domains: Some Simple Observations**  
- Some Observations on Some Attribute Values  
- Physics Attributes  
  - SI: The International System of Quantities  
- What Are We to Learn from this Exposition?  
- Attribute Types, Scales and Values: Some Thoughts  

**Conclusion**  
- What Have We Achieved?  
- Domain Descriptions as Basis for Requirements Prescriptions  
- What Next?  
- Thanks  

**Bibliographical Notes**  

**References to Draft Domain Descriptions**  

**References**
1. **Introduction**

1.1. **On the Issues of Implicit and Explicit Semantics**

- In [2] the issues of implicit and explicit semantics are analysed.

- It appears, from [2], that when an issue
  - of software requirements or
  - of the context, or, as we shall call it, the domain,
  - is not prescribed or described
  - to the extent that is relied upon in the software design,
  - then it is referred to as an issue of implicit semantics.

- Once prescribed, respectively described,
  that issue becomes one of explicit semantics.

- In this invited talk I offer
  
  **a calculus for analysing & describing domains**
  
  **a calculus that allows you**
  
  **to systematically and formally describe domains.**
1.2. A Triptych of Software Engineering

- The dogma is:
  - before software can be designed
    - we must understand its requirements;
  - and before we can prescribe the requirements
    - we must understand the domain,
  - that is, describe the domain.
• A strict, but not a necessary, interpretation of this dogma thus suggests that software development “ideally” proceeds in three phases:

- First a phase of **domain engineering** in which an analysis of the application domain leads to a description of that domain.\(^1\)
- Then a phase of **requirements engineering** in which an analysis of the domain description leads to a prescription of requirements to software for that domain.
- And, finally, a phase of **software design** in which an analysis of the requirements prescription leads to software for that domain.

\(^1\)This phase is often misunderstood. On one hand we expect domain stakeholders, e.g., bank associations and university economics departments, to establish “a family” of bank domain descriptions: taught when training and educating new employees, resp. students. Together this ‘family’ covers as much as is known about banking. On the other hand we expect each new bank application (software) development to “carve” out a “sufficiently large” description of the domain it is to focus on. Please replace the term bank with an appropriate term for the domain for which You are to develop software.
• Proof of program, i.e., software code, correctness can be expressed as:
  $D, S \models R$
  which we read as:
  proofs that $S$oftware
  is correct with respect to $R$equirements
  implies references to the $D$omain.

- Often the domain is referred to as the **context**.
- We treat contexts, i.e., domain descriptions as first class citizens [2, Abstract, Page 1, lines 9–10].
- By emphasizing the formalisation of domain descriptions we thus focus on the **explicit** semantics.
- Our approach, [1], summarised in Sect. 2. of this paper, thus represents a formal approach to the description of contexts (i.e., domains) [2, Abstract, Page 1, line 12].
- By a **domain**, i.e., a context, **description**, we shall here understand an **explicit semantics** of what is usually not specified and, when not so, referred to as **implicit semantics**².

²“The contexts . . . are treated as second-class citizens: in general, the modelling is implicit and usually distributed
1.4. **Semantics**

- I use the term ‘semantics’ rather than the term ‘knowledge’.
- The reason is this:
  - The entities are what we can meaningfully speak about.
    - That is, the names of the endurants and perdurants,
    - of their being atomic or composite, discrete or continuous,
    - parts, components or materials,
    - their unique identifications, mereologies and attributes,
    - and the types, values and use of operations over these,
    - form the language spoken by practitioners in the domain.
- It is this language
  - its base syntactic quantities and
  - semantic domains
  - we structure and ascribe a semantics.

between the requirements model and the system model.” [2, Abstract, Page 1, lines 9–12].
1.5. Method & Methodology

• By a **method** I understand
  ❚ a set of principles
  ❚ for selecting and applying
  ❚ techniques and tools
    for constructing a manifest or an abstract artifact.

• By **methodology** I understand the study and knowledge of methods.

• **My work is almost exclusively in the area of methods and methodology.**
1.6. Computer & Computing Sciences

- By **computer science** I understand
  - the study and knowledge about the things
  - that can exist inside computing devices.

- By **computing science** I understand
  - the study and knowledge about how to construct the things
  - that can exist inside computing devices.

Computing science is also often referred to as **programming methodology**.

- **My work is almost exclusively in the area of computing science.**
2. The Analysis & Description Prompts

- We present a calculus of analysis and description prompts\(^3\).
- The presentation here is a very short version of [1, Sects. 2–4, 31 pages].
  - These prompts are tools that the domain analyser & describer uses.
  - The domain analyser & describer is in the domain, sees it, can touch it, and then applies the prompts, in some orderly fashion, to what is being observed.
  - So, on one hand, there is the necessarily informal domain, and, on the other hand, there are the seemingly formal prompts and the "suggestions for something to be said", i.e., written down: narrated and formalised.

---

\(^3\) Prompt, as a verb: to move or induce to action; to occasion or incite; inspire; to assist (a person speaking) by "suggesting something to be said".
2. The Analysis & Description Prompts

An Ontology for Manifest Domains
Basic Categories and their Relations

"a thing"

Describable

is_entity

is_perdurant

is_endurant

is_continuous

is_discrete

is_material

is_atomic

is_part

has_concrete_type

obs_part_type

has_materials

obs_materials

endurants: external qualities

endurants: internal qualities

has_components

obs_part_sorts

obs_comp_sorts

has_mereology

obs_mereology

unique identifiers

obs_uid

obs_attributes

attribute categories

analytic prompts

description prompts

attribute categories

obs_... is an abbreviation for ‘observe’
The Analysis & Description Prompts

- The figure suggests a number of analysis and description prompts.
  - The domain analyser & describer is “positioned” at the top.
  - If what is observed can be conceived and described then it is \textbf{an entity}.
  - If it can be described as a “complete thing” at no matter which given snapshot of time then it is \textbf{an endurant}.
  - If it is an entity but for which only a fragment exists if we look at or touch them at any given snapshot in time, then it is \textbf{a perdurant}.
2.1. **Endurants: Parts, Components and Materials**

- Endurants are either **discrete** or **continuous**.
  - With discrete endurants we can choose to associate, or to not associate *mereologies*\(^4\).
  - If we do we shall refer to them as **parts**,
  - else we shall call them **components**.
- With continuous endurants we do not associate mereologies.
- The continuous endurants we shall also refer to as *(gaseous or liquid)* **materials**.

- Parts are either **atomic** or **composite** and all parts have
  - *unique identifiers*,
  - *mereology* and
  - *attributes*.

---

\(^4\) — 'mereology' will be explained next
2. The Analysis & Description Prompts 2.1. Endurants: Parts, Components and Materials

- If the observed part, \( p:P \), is \textit{is\_composite}
  
  ✷ then we can observe the part sorts and values, \( P_1, P_2, ..., P_m \) respectively \( p_1, p_2, ..., p_m \) of \( p \).
  
  ✷ “Applying” \texttt{observe\_part\_sorts} to \( p \) yields
    
    ✷ an informal (i.e., a \texttt{narrative}) and
    
    ✷ a \textbf{formal} description:
2. The Analysis & Description Prompts 2.1. Endurants: Parts, Components and Materials

**Schema:** Composite Parts

- **Narrative:**
  - ...

- **Formal:**
  - **type**
    - $P_1, P_2, ..., P_m$
  - **value**
    - $\text{obs}_{P_i}: P \rightarrow P_i$

repeated for all $m$ part sorts $P_i$s
Aircraft Example 1: The Pragmatics

- The **pragmatics**\(^5\) of this ongoing example is this:
  - We are dealing with ordinary passenger aircraft.
  - We are focusing on that tiny area of concern that focus on passengers being informed of the progress of the flight, once in the air:
    - where is the aircraft:
      - its current position somewhere above the earth;
      - its current speed and direction
      - and possible acceleration (or deceleration);
      - We do not bother about what time it is – etc.
      - We abstract from the concrete presentation of this information.

\(^5\)Pragmatics is here used in the sense outlined in [4, Chapter 7, Pages 145–148].
Aircraft Example 2: Parts

1 An *aircraft* is composed from several parts of which we focus on
   a a *position* part,
   b a *travel dynamics* part, and
   c a *display* part.

type
1 AC, PP, TD, DP
value
1a obs_PP: AC → PP
1b obs_TD: AC → TD
1c obs_DP: AC → DP
• We have just summarised the analysis and description aspects of endurants in *extension* (their “form”).

• We now summarise the analysis and description aspects of endurants in *intension* (their “contents”).

• There are three kinds of intensional *qualities* associated with parts, two with components, and one with materials.
  ◦ Parts and components, by definition, have *unique identifiers*;
  ◦ parts have *mereologies*,
  ◦ and all endurants have *attributes*. 
2.2. Internal Qualities

2.2.1. Unique Identifiers

- Unique identifiers are further undefined tokens that uniquely identify parts and components.
- The description language observer \texttt{uid\_P}, when applied to parts \( p: P \) yields the unique identifier, \( \pi: \Pi \), of \( p \).
- So the \texttt{observe\_part\_sorts}(p) invocation also yields the description text:
The Analysis & Description Prompts

2. Internal Qualities

2.2. Unique Identifiers

---

**Schema: Unique Identifiers**

- ... [added to the narrative and]
- **type**
  - $\Pi_1, \Pi_2, \ldots, \Pi_m$
- **value**
  - $\text{uid}_{\Pi_i} : P_i \rightarrow \Pi_i$

repeated for all $m$ part sorts $P_i$s and added to the formalisation.
Example 3: Unique Identifiers

2 position, travel dynamic and display parts have unique identifiers.

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPI, TDI, DPI</td>
<td>uid_PP: PP → PPI</td>
</tr>
<tr>
<td></td>
<td>uid_TD: TD → TDI</td>
</tr>
<tr>
<td></td>
<td>uid_DP: DP → DPI</td>
</tr>
</tbody>
</table>
2.2.2. Mereology

- **Mereology is the study and knowledge of parts and part relations.**
  - The mereology of a part is an expression over the unique identifiers of the (other) parts with which it is related,
  - hence **mero_P**: $P \rightarrow \mathcal{E}(\Pi_j, ..., \Pi_k)$ where $\mathcal{E}(\Pi_j, ..., \Pi_k)$ is a type expression.
  - So the **observe_part_sorts**($p$) invocation also yields the description text:
2. The Analysis & Description Prompts 2.2. Internal Qualities 2.2.2. Mereology

**Schema: Mereology**

- ... [added to the narrative and]
- **value**
  - mereo\_Pi : Pi → E_i(Π_{i,j}, ..., Π_{i,k}) [added to the formalisation]
Aircraft **Example 4: Mereology**

- We shall omit treatment of aircraft mereologies.

3 The position part is related to the display part.

4 The travel dynamics part is related to the display part.

5 The display part is related to both the position and the travel dynamics parts.

<table>
<thead>
<tr>
<th>Value</th>
<th>mereo_PP: PP → DPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>mereo_TD: TP → DPI</td>
</tr>
<tr>
<td>4</td>
<td>mereo_DP: DP → PPI×TDI</td>
</tr>
</tbody>
</table>
2.2.3. Attributes

• Attributes are the remaining qualities of endurants.
  ✧ The analysis prompt `obs_attributes` applied to an endurant
    yields a set of type names, $A_1, A_2, ..., A_t$, of attributes.
  ✧ They imply the additional description text:
2. The Analysis & Description Prompts

2.2. Internal Qualities

2.2.3. Attributes

---

**Schema:** Attributes

- **Narrative:**
  - ...

- **Formal:**
  - **type**
    - $A_1, A_2, \ldots, A_t$
  - **value**
    - attr_{A_i} : $E \rightarrow A_i$

repeated for all $t$ attribute sorts $A_i$s!
Aircraft **Example 5:** Position Attributes

6 Position parts have longitude, latitude and altitude attributes.

**type**

6 LO, LA, AL

**value**

6 attr_LO: PP $\rightarrow$ LO
6 attr_LA: PP $\rightarrow$ LA
6 attr_AL: PP $\rightarrow$ AL

- These quantities: longitude, latitude and altitude
  - are “actual” quantities, they mean what they express,
  - they are not recordings or displays of these quantities;
  - to express those we introduce separate types.
Aircraft Example 6: Travel Dynamics Attributes

Travel dynamics parts have velocity\(^6\) and acceleration\(^7\).

- These quantities: velocity and acceleration,
  - are "actual" quantities, they mean what they express,
  - they are not recordings or displays of these quantities;
  - to express those we introduce separate types.

---

\(^6\)Velocity is a vector of speed and orientation (i.e., direction)

\(^7\)Acceleration is a vector of change of speed per time unit and orientation.
Aircraft **Example 7: Quantity Recordings**

8 On one hand there are the actual location and dynamics quantities (i.e., values),

9 on the other hand there are their recodings,

10 and there are conversion functions from actual to recorded values.

**type**
8  LO, LA, AL, VEL, ACC
9  rLO, rLA, rAL, rVEL, rACC

**value**
10  a2rLO: LO → rLO, a2rLA: LA → rLA, a2rAL: AL → rAL
10  a2rVEL: VEL → rVEL, a2rACC: ACC → rACC

- There are, of course, no functions that convert recordings to actual values!
Aircraft **Example 8**: Display Attributes

11 Display parts have display modified longitude, latitude and altitude, and velocity and acceleration attributes – with functions that convert between these, recorded and displayed, attributes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>dLO, dLA, dAL</td>
<td>11 attr_dLO: DP → dLO</td>
</tr>
<tr>
<td>dVEL, dACC</td>
<td>11 attr_dLA: DP → dLA</td>
</tr>
<tr>
<td></td>
<td>11 attr_dAL: DP → dAL</td>
</tr>
<tr>
<td></td>
<td>11 attr_dVEL: DP → dVEL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Conversion</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r2dLO, d2rLO</td>
<td>11 r2dLO, d2rLO: rLO ↔ dLO</td>
</tr>
<tr>
<td>r2dLA, d2rLA</td>
<td>11 r2dLA, d2rLA: rLA ↔ dLA</td>
</tr>
<tr>
<td>r2dAL, d2rAL</td>
<td>11 r2dAL, d2rAL: rAL ↔ dAL</td>
</tr>
<tr>
<td>r2dVEL, d2rVEL</td>
<td>11 r2dVEL, d2rVEL: rVEL ↔ dVEL</td>
</tr>
<tr>
<td>r2dACC, d2rACC</td>
<td>11 r2dACC, d2rACC: rACC ↔ dACC</td>
</tr>
</tbody>
</table>

**Axiom**

∀ rlo: rLO • d2rLO(r2dLO(rlo)) = rlo etcetera!
2.2.4. Attribute Categories

• Michael A. Jackson [5] categorizes and defines attributes as either
  ◇ static or
  ◇ dynamic,

• with dynamic attributes being either
  ◇ inert,
  ◇ reactive or
  ◇ active.

• The latter are then either
  ◇ autonomous,
  ◇ biddable or
  ◇ programmable.

• This categorization has a strong bearing on how these (f.ex., part) attributes are dealt with when now interpreting parts as behaviours.
Aircraft **Example 9**: Attribute Categories

12 Longitude, latitude, altitude, velocity and acceleration are all reactive attributes – they change in response to the bidding of aircraft attributes that we have not covered\(^8\).

13 Their display modified forms are all programmable attributes.

**attribute categories**

12 *reactive*: LO, LA, AL, VEL, ACC

13 *programmable*: dLO, dLA, dAL, dVEL, dACC

\(^8\) for example: *thrust*, *weight*, *lift*, *drag*, *rudder position*, and *aileron position* – plus dozens of other attributes
2.3. Description Axioms and Proof Obligations

• In [1] we show that the description prompts may result in axioms or proof obligations.

  ⦿ We refer to [1] for details.
  ⦿ Here we shall, but show one example of an axiom.
Aircraft Example 10: An Axiom

The displayed attributes must at any time be displayings of the corresponding recorded position and travel dynamics attributes.

Axiom

\[ \square \forall \text{ac:AC} \cdot \]

1. Let \((pp,td,di) = (\text{obs}_{PP}(ac), \text{obs}_{TD}(ac), \text{obs}_{DP}(ac))\) in
2. Let \((lo,la,at) = (\text{attr}_{LO}(pp), \text{attr}_{LA}(pp), \text{attr}_{AT}(pp))\),
3. \((\text{vel}, \text{acc}, \text{dir}) = (\text{attr}_{VEL}(td), \text{obs}_{ACC}(td))\),
4. \((dlo,dla,dat) = (\text{attr}_{dLO}(di), \text{attr}_{dLA}(di), \text{attr}_{dAT}(di))\),
5. \((\text{dvel}, \text{dacc}) = (\text{attr}_{dVEL}(di), \text{obs}_{dACC}(di))\) in
6. \((\text{dlo}, \text{dla}, \text{dat}) = (r2dLO(a2rLO(lo)), r2dLA(a2rLA(la)), r2dAL(a2rAL(at)))\)
7. \((\text{dvel}, \text{dacc}) = (r2dVEL(a2rVEL(vel)), r2dACC(a2rACC(acc)))\)
8. end end
2.4. From Manifest Parts (Endurants) to Domain Behaviours (Perdurants)

- [1] then presents a *compiler* which to manifest *parts* associate *behaviours*.

- These are then specified as CSP [6] *processes*.

### 2.4.1. The Idea — by means of an example

- The term *aircraft* can have the following “meanings”:
  - the *aircraft*, as an *endurant*,
    parked at the airport gate,
    i.e., as a *composite part*;
  - the *aircraft*, as a *perdurant*,
    as it flies through the skies,
    i.e., as a *behaviour*; and
  - the *aircraft*, as an *attribute*,
    of an airline timetable.
2. The Analysis & Description Prompts

2.4. From Manifest Parts (Endurants) to Domain Behaviours (Perdurants)

2.4.1. The Idea — by means of an example

Aircraft **Example 11:** An Informal Story I/II

- An aircraft has the following behaviours:
  - the *position* behaviour;
    - it observes the aircraft location attributes: *longitude*, *latitude* and *altitude*;
    - record and communicate these, as a triple, to the *display* behaviour;
  - the *travel dynamics* behaviour;
    - it observes the aircraft travel dynamics attributes *velocity* and *acceleration*;
    - record and communicate these, as a triple, to the *display* behaviour; and
  - the *display* behaviour receives two tuplets of attribute value recordings from respective *position* and *travel dynamics* behaviours and display these recorded attribute values: *longitude*, *latitude*, *altitude*, *velocity* and *acceleration* in some form.
The six actual **position** and **travel dynamics** attribute values

*longitude, latitude, altitude, velocity* and *acceleration*

*are recorded*, by appropriate instruments.

*In the above figure this is indicated by input channels*

`attr_LO_ch, attr_LA_ch, attr_AL_ch, attr_VEL_ch` and `attr_ACC_ch`. 
2.4.2. Channels and Communication

- Behaviours sometimes synchronise and usually communicate.
- We use the CSP [6] notation (adopted by RSL) to model behaviour communication.
- Communication is abstracted as the sending,
  - $\texttt{ch} ! \texttt{m}$, and receipt,
  - $\texttt{ch} ?$, of messages,
  - $\texttt{m:M}$, over channels,
  - $\texttt{ch}$.

  **type M**
  **channel ch:M**
Aircraft **Example 12**: Channels I/II

15 The messages sent from the *position* behaviour to the *display* behaviour are triplets of recorded longitude, latitude and altitude values.

16 The messages sent from the *travel dynamics* behaviour to the *display* behaviour are duplets of recorded velocity and acceleration values.

17 There is a channel, `po_di_ch`, that allows communication of messages from the position behaviour to the display behaviour.

18 There is a channel, `td_di_ch`, that allows communication of messages from the travel dynamics behaviour to the display behaviour.

19 For each of the reactive attributes there is a corresponding channel.
Aircraft **Example 12**: Channels II/II

<table>
<thead>
<tr>
<th>Type</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM</td>
<td>$r_{LO} \times r_{LA} \times r_{AL}$</td>
</tr>
<tr>
<td>TDM</td>
<td>$r_{VEL} \times r_{ACC}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>po_di_ch:PM</td>
</tr>
<tr>
<td>td_di_ch:TDM</td>
</tr>
<tr>
<td>attr_LO_ch:LO, attr_LA_ch:LA, attr_AL_ch:AL</td>
</tr>
<tr>
<td>attr_VEL_ch:VEL, attr_ACC_ch:ACC</td>
</tr>
</tbody>
</table>
2.4.3. Behaviour Signatures

- We shall only cover behaviour signatures when expressed in RSL/CSP [7].
- The behaviour functions are now called processes.
- That a behaviour function is a never-ending function, i.e., a process, is “revealed” in the function signature by the “trailing” Unit:
  \[
  \text{behaviour}: \ldots \rightarrow \ldots \text{Unit}
  \]
- That a process takes no argument is ”revealed” by a “leading” Unit:
  \[
  \text{behaviour}: \text{Unit} \rightarrow \ldots
  \]
- That a process accepts channel, viz.: \text{ch}, inputs, including accesses an external attribute \text{A}, is “revealed” in the function signature as follows:
  \[
  \text{behaviour}: \ldots \rightarrow \text{in ch} \ldots \text{, resp. in attr_A_ch}
  \]
• That a process offers channel, viz.: \texttt{ch}, outputs is “revealed” in the function signature as follows:

\[
\text{behaviour: \ldots \rightarrow \texttt{out} \ ch \ \ldots}
\]

• That a process accepts other arguments is “revealed” in the function signature as follows:

\[
\text{behaviour: ARG \rightarrow \ldots}
\]

• where \texttt{ARG} can be any type expression:

\[
T, \ T \rightarrow T, \ T \rightarrow T \rightarrow T, \ \text{etcetera}
\]

where \( T \) is any type expression.
2.4.4. Translation of Part Qualities

- Part qualities, that is: *unique identifiers, mereologies* and *attributes*, are translated into behaviour arguments – of one kind or another, i.e., (...).

  ✷ Typically we can choose to *index* behaviour names, \( b \) by the *unique identifier*, \( id \), of the part based on which they were translated, i.e., \( b_{id} \).

  ✷ *Mereology values* are usually static, and can, as thus, be treated like we treat static attributes (see next), or can be set by their behaviour, and are then treated like we treat programmable attributes (see next), i.e., (...).
静态属性 (Static attributes) 变成行为定义 (body) 的常数值。

惰性、反应性和自主属性 (Inert, reactive and autonomous attributes) 变成对通道的引用，例如 `ch_dyn`，这样当需要惰性、反应性和自主属性的值时，它会表达为 `ch_dyn ?`。

可编程和可委派属性 (Programmable and biddable attributes) 变成传入行为的尾递归调用的参数，并且可能在行为定义的体内更新，即，(...).
2.4.5. Part Behaviour Signatures

- We can, without loss of generality, associate with each part a behaviour;
  - parts which share attributes
  - (and are therefore referred to in some parts’ mereology),
  - can communicate (their “sharing”) via channels.
A behaviour signature is therefore:

\[
\text{beh}_{\pi: \Pi}: \text{me:MT} \times \text{sa:SA} \rightarrow \text{ca:CA} \rightarrow \text{in ichns(ea:EA)} \text{ in, out iochs(me)} \text{ Units}
\]

where

- (i) \(\pi: \Pi\) is the unique identifier of part \(p\), i.e., \(\pi = \text{uid}_P(p)\),
- (ii) \(\text{me:ME}\) is the mereology of part \(p\), \(\text{me} = \text{obs_mereo}_P(p)\),
- (iii) \(\text{sa:SA}\) lists the static attribute values of the part,
- (iv) \(\text{ca:CA}\) lists the biddable and programmable attribute values of the part,
- (v) \(\text{ichns(ea:EA)}\) refer to the external attribute input channels, and where
- (vi) \(\text{iochs(me)}\) are the input/output channels serving the attributes shared between the part \(p\) and the parts designated in its mereology \(\text{me}\).
Aircraft Example 13: Part Behaviour Signatures, I/II

20 The signature of the *position* behaviour lists its unique identifier, mereology, no static and no controllable attributes, but its three reactive attributes (as input channels) and its (output) channel to the *display* behaviour.

21 The signature of the *travel dynamics* behaviour lists its unique identifier, mereology, no static and no controllable attributes, but its three reactive attributes (as input channels) and its (output) channel to the *display* behaviour.

22 The signature of the *display* behaviour lists its unique identifier, its mereology, no static attribute, but the programmable display attributes, assembled in a pair of a triplet and duplet, and its two input channels from the *position*, respectively the *travel dynamics* behaviours.
Aircraft Example 14: Part Behaviour Signatures, I/II

type
22 $DA = (dLA \times dLO \times dAL) \times (dVEL \times dACC)$

value
20 position: $PI \times DPI \rightarrow$
20 $\text{in} \ attr_{LO\_ch}, attr_{LA\_ch}, attr_{AL\_ch}, \text{out} \ po_{di\_ch} \ Unit$
21 travel_dynamics: $TDI \times DPI \rightarrow$
21 $\text{in} \ attr_{VEL\_ch}, attr_{ACC\_ch}, attr_{DIR\_ch}, \text{out} \ td_{di\_ch} \ Unit$
22 display: $DI \times (PPI \times TDI) \rightarrow DA \rightarrow \text{in} \ po_{di\_ch}, td_{di\_ch} \ Unit$
2.4.6. Behaviour Compilations

2.4.6.1 Composite Behaviours

- Let $P$ be a composite sort defined in terms of sub sorts $P_1, P_2, \ldots, P_n$.

  - The process definition compiled from $p:P$, is composed from
    - a process description, $McP_{uid\ P}(p)$, relying on and handling
      the unique identifier, mereology and attributes of part $p$
    - operating in parallel with processes $p_1, p_2, \ldots, p_n$ where
      * $p_1$ is compiled from $p_1:P_1$,
      * $p_2$ is compiled from $p_2:P_2$,
      * $\ldots$, and
      * $p_n$ is compiled from $p_n:P_n$.

- The domain description “compilation” schematic below
  “formalises” the above.
Process Schema: Abstract is_composite(p)

\[
\text{value} \\
\text{compile_process: P } \rightarrow \text{ RSL-Text} \\
\text{compile_process}(p) \equiv \\
\mathcal{M}P_{\text{uid}_P(p)}(\text{obs\_mero}_P(p), S_A(p))(C_A(p)) \\
\| \text{compile_process}(\text{obs\_part}_P_1(p)) \\
\| \text{compile_process}(\text{obs\_part}_P_2(p)) \\
\| \ldots \\
\| \text{compile_process}(\text{obs\_part}_P_n(p))
\]

- The text macros: $S_A$ and $C_A$ were informally explained above.
- Part sorts $P_1$, $P_2$, ..., $P_n$ are obtained from the observe_part_sorts prompt.
Aircraft Example 15: Aircraft Behaviour, I/II

23 Compiling a composite aircraft part results in the parallel composition

   a the compilation of the atomic position part,
   b the compilation of the atomic travel dynamics part, and
   c the compilation of the atomic display part.

We omit compiling the aircraft core behaviour.

24 Compilation of atomic parts entail no further compilations.
## Aircraft Example 15: Aircraft Behaviour, II/II

<table>
<thead>
<tr>
<th>value</th>
<th>compile(ac) ≡</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>compile(obs_PP(p))</td>
</tr>
<tr>
<td>23a</td>
<td>compile(obs_TD(p))</td>
</tr>
<tr>
<td>23b</td>
<td>compile(obs_DL(p))</td>
</tr>
</tbody>
</table>
2.4.6.2 Atomic Behaviours

Process Schema: \( \text{is\_atomic}(p) \)

\[
\begin{align*}
\text{value} & \\
\text{compile\_process: } P & \rightarrow \text{RSL-Text} \\
\text{compile\_process}(p) & \equiv \\
\mathcal{MP}_{\text{uid}_P(p)}^{\text{obs\_merelo}_P(p), \mathcal{S}_A(p), \mathcal{C}_A(p)}
\end{align*}
\]
Aircraft Example 16: Atomic Behaviours

25 We initialise the display behaviour with a further undefined value.

value

23a \( \text{compile}(\text{obs\_PP}(p)) \equiv \)
23a \( \text{position}(\text{uid\_PP}(p),\text{mereo\_PP}(p)) \)
23b \( \text{compile}(\text{obs\_TD}(p)) \equiv \)
23b \( \text{travel\_dynamics}(\text{uid\_TD}(p),\text{mereo\_TD}(p)) \)
25 \( \text{init\_DA}\!:\!\text{DA} = \ldots \)
23c \( \text{compile}(\text{obs\_DI}(p)) \equiv \)
23c \( \text{display}(\text{uid\_DI}(p),\text{mereo\_DI}(p))(\text{init\_DA}) \)

- In the above we have already subsumed the *atomic behaviour definitions*, see next, and directly inserted the \( \mathcal{F} \) definitions.
2.4.7. Atomic Behaviour Definitions

<table>
<thead>
<tr>
<th>Process Schema IV: Atomic Core Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
</tr>
<tr>
<td>( MP_{\pi:}\Pi: me:MT \times sa:SA \rightarrow ca:CA \rightarrow )</td>
</tr>
<tr>
<td>( \text{in ichns}(ea:EA) \text{ in, out iochs}(me) \text{ Unit} )</td>
</tr>
<tr>
<td>( MP_{\pi:}\Pi(me,sa)(ca) \equiv )</td>
</tr>
<tr>
<td>( \text{let } (me',ca') = F_{\pi:}\Pi(me,sa)(ca) \text{ in, out} )</td>
</tr>
<tr>
<td>( MP_{\pi:}\Pi(me',sa)(ca') \text{ end} )</td>
</tr>
</tbody>
</table>

\[
F_{\pi:}\Pi: me:MT \times sa:SA \rightarrow CA \rightarrow \text{ in ichns}(ea:EA) \text{ in, out iochs}(me) \rightarrow MT \times CA
\]
Aircraft Example 17: Position Behaviour Definition

26 The *position* behaviour offers to receive
   the *longitude*, *latitude* and the *altitude* attribute values
27 and to offer them to the *display* behaviour,
28 whereupon it resumes being the *position* behaviour.

value
20  position(p\pi,d\pi) \equiv
26    let (lo,la,al) = (attr_LO_ch?,attr_LA_ch?,attr_AL_ch?) in
27    po_di_ch ! (a2rLO(lo),a2rLA(la),a2rAL(al)) ;
28    position(p\pi,d\pi) end
Aircraft **Example 18:** Travel Dynamics Behaviour Definition

29 The `travel_dynamics` behaviour offers to receive
the recorded *velocity* and the *acceleration* attribute values
30 and to offer these to the `display` behaviour,
31 whereupon it resumes being the `travel_dynamics` behaviour.

\[
\text{travel_dynamics}(\text{td} \pi, \text{d} \pi) \equiv \\
\text{let } (\text{vel}, \text{acc}) = (\text{attr}_{\text{VEL}}\_\text{ch}?, \text{attr}_{\text{ACC}}\_\text{ch}? ) \text{ in } \\
\text{td}\_\text{di}\_\text{ch} ! (\text{a2rVEL}(\text{vel}), \text{a2rACC}(\text{acc})) ; \\
\text{travel_dynamics}(\text{td} \pi, \text{d} \pi) \text{ end }
\]
2. The Analysis & Description Prompts 2.4. From Manifest Parts (Endurants) to Domain Behaviours (Perdurants) 2.4.7. Atomic Behaviour Definitions

---

**Aircraft Example 19: Display Behaviour Definition**

32 The *display* behaviour offers
   to receive the reactive attribute tuplets
   from the *position* and the *travel_dynamics* behaviours while

33 resuming to be that behaviour albeit now with these
   as their updated display.

34 The *conv*ersion functions are extensions of the ones introduced earlier.

```plaintext
value
22 display(dπ,(dπ,tdπ))(d_pos,d_tdy) ≡
32 let (pos_d’,tdy_d’) = (po_di_ch?,td_di_ch?) in
33 display(dπ,(dπ,tdπ))(conv(pos_d’),conv(c_tdy_d’)) end

type
34 dMPD = dLO × dLA × dAL
34 dMTD = dVEL × dACC

value
34 conv: MPD → dMPD
34 conv(rlo,rla,ral) ≡ (r2dLO(rlo),r2dLA(rla),r2dAL(ral))
34 conv: MTD → dMTD
34 conv(rvel,racc) ≡ (r2dVEL(rvel),r2dACC(racc))
```
2.5. A Proof Obligation

• We refer, again, to [1] for more on proof obligations.

Aircraft Example 20: A Proof Obligation

• The perdurant descriptions of Items 15–34
• is a model of the axiom expressed in Item 14.

- This section of the talk covers three loosely related topics:
  - First we muse over properties of some attribute values.
  - Then we recall some facts about types, scales and values of measurable units in physics.
  - The previous leads us to consider further detailing the concept of attributes such as we have covered it in Sect. 2.2.3, Slides 29–36, and in [1].

- The reason for covering these topics is that
  - most attribute values are represented in “final” programs as numbers of one kind or another
  - and that type checking in most software is with respect to these numbers.
3.1. **Some Observations on Some Attribute Values**

Let us, seemingly randomly, examine some simple, e.g., arithmetic, operations in classical domains.

- By *time* is often meant absolute time.
- So a time could be *November 16, 2017: 00:23 am*.
- One can not add two times.
- One can speak of a time being earlier, or before another time.
- *October 23, 2017: 10:01 am* is earlier, \( \leq \), than *November 16, 2017: 00:23 am*.
- One can speak of the time interval between
  - *October 23, 2016: 8:01 am* and *October 24, 2017: 10:05 am*
  - being *1 year, 1 day, 2 hours and 4 minutes*, that is: *October 24, 2017: 10:05 am \( \ominus \) October 23, 2016: 8:01 am = 1 year, 1 day, 2 hours and 4 minutes*
• One can add a *time interval* to a *time* and obtain a *time*.
• One can multiply a *time interval* with a *real*\(^9\)
• We can formalize the above:

**type**

\[
\begin{align*}
\mathbb{T} &= \text{Month} \times \text{Day} \times \text{Year} \times \text{Hour} \times \text{Minute} \times \text{Sec} \ldots \\
\mathbb{TII} &= \text{Days} \times \text{Hours} \times \text{Minutes} \times \text{Seconds} \times \ldots \\
\text{Month} &= \{1,2,3,4,5,6,7,8,9,10,11,12\} \\
\text{Day} &= \{1,2,3,4,\ldots,28,29,30,31\} \\
\text{Hour,Hours} &= \{0,1,2,3,\ldots,21,22,23\} \\
\text{Minute,Minutes} &= \{0,1,2,3,\ldots,56,57,58,59\} \\
\text{Second,Seconds} &= \{0,1,2,3,\ldots,56,57,58,59\} \\
\ldots
\end{align*}
\]

\[
\begin{align*}
\text{Days} &= \text{Nat}
\end{align*}
\]

\(^9\)The time interval could, e.g., be converted into seconds, then the integer number standing for seconds can be multiplied by \(r\) and the result be converted “back” into years, days, hours, minutes and seconds — whatever it takes!
value

\[<, \leq, =, \geq, > : T \times T \rightarrow \text{Bool}\]
\[- : T \times T \rightarrow TI\]
\[\text{pre} t - t' : t' \leq t\]
\[<, \leq, =, \geq, > : TI \times TI \rightarrow \text{Bool}\]
\[-, + : TI \times TI \rightarrow TI\]
\[\ast : TI \times \text{Real} \rightarrow TI\]
\[/ : TI \times TI \rightarrow \text{Real}\]
• One can not add temperatures – makes no sense in physics!
  ◇ But one can take the mean value of two (or more) temperatures.
  ◇ One can subtract temperatures
    obtaining positive or negative temperature intervals.
  ◇ One can take the mean of any number of temperature,
    but would probably be well advised to have these represent
    regular sampling, or at least time-stamped.
  ◇ One can also define \textit{rate of change of temperature}. 
type
  Temp, MeanTemp, Degrees, TempIntv = Degrees
value
  mean: Temp-set × Nat → MeanTemp
  −: Temp × Temp → TempIntv
type
  TST = (Temp × ℤ)-set
value
  avg: TST → MeanTemp
type
  TimeUnit = {"year", "month", "day", "hour", ...}
  RoTC = TempIntv × TimeUnit

• Etcetera.
• We leave it to the listener to speculate on which operations one can perform on a person’s attributes: height, weight, birth date, name, etc.

• And similarly for other domains.

• It is time to “lift” these observations.
  
  ✤ After the examples above we should inquire as to which kind of units we may operate upon.
  
  ✤ For the sake of our later exposition it is enough that we look in some detail at the “universe” of physics.

3.2. Physics Attributes

3.2.1. SI: The International System of Quantities

- In physics we operate on values of attributes of manifest, i.e., physical phenomena.
- The type of some of these attributes are recorded in well known tables, cf. Tables 1–3.
Table 1 shows the base units of physics.

<table>
<thead>
<tr>
<th>Base quantity</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>meter</td>
<td>m</td>
</tr>
<tr>
<td>mass</td>
<td>kilogram</td>
<td>kg</td>
</tr>
<tr>
<td>time</td>
<td>second</td>
<td>s</td>
</tr>
<tr>
<td>electric current</td>
<td>ampere</td>
<td>A</td>
</tr>
<tr>
<td>thermodynamic temperature</td>
<td>kelvin</td>
<td>K</td>
</tr>
<tr>
<td>amount of substance</td>
<td>mole</td>
<td>mol</td>
</tr>
<tr>
<td>luminous intensity</td>
<td>candela</td>
<td>cd</td>
</tr>
</tbody>
</table>

Table 1: Base Units
Table 2 shows the units of physics derived from the base units.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Derived Quantity</th>
<th>Derived Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>radian</td>
<td>rad</td>
<td>angle</td>
<td>m/m</td>
</tr>
<tr>
<td>steradian</td>
<td>sr</td>
<td>solid angle</td>
<td>m²×m⁻²</td>
</tr>
<tr>
<td>Hertz</td>
<td>Hz</td>
<td>frequency</td>
<td>s⁻¹</td>
</tr>
<tr>
<td>newton</td>
<td>N</td>
<td>force, weight</td>
<td>kg×m×s⁻²</td>
</tr>
<tr>
<td>pascal</td>
<td>Pa</td>
<td>pressure, stress</td>
<td>N/m²</td>
</tr>
<tr>
<td>joule</td>
<td>J</td>
<td>energy, work, heat</td>
<td>N×m</td>
</tr>
<tr>
<td>watt</td>
<td>W</td>
<td>power, radiant flux</td>
<td>J/s</td>
</tr>
<tr>
<td>coulomb</td>
<td>C</td>
<td>electric charge</td>
<td>s×A</td>
</tr>
<tr>
<td>volt</td>
<td>V</td>
<td>voltage, electromotive force</td>
<td>W/A (kg×m²×s⁻³×A⁻¹)</td>
</tr>
<tr>
<td>farad</td>
<td>F</td>
<td>capacitance</td>
<td>C/V (kg⁻¹×m⁻²×s⁴×A²)</td>
</tr>
<tr>
<td>ohm</td>
<td>Ω</td>
<td>electrical resistance</td>
<td>V/A (kg×m²×s³×A²)</td>
</tr>
<tr>
<td>siemens</td>
<td>S</td>
<td>electrical conductance</td>
<td>A/V (kg¹×m²×s³×A²)</td>
</tr>
<tr>
<td>weber</td>
<td>Wb</td>
<td>magnetic flux</td>
<td>V×s (kg×m²×s⁻²×A⁻¹)</td>
</tr>
<tr>
<td>tesla</td>
<td>T</td>
<td>magnetic flux density</td>
<td>Wb/m² (kg×s²×A⁻¹)</td>
</tr>
<tr>
<td>henry</td>
<td>H</td>
<td>inductance</td>
<td>Wb/A (kg×m²×s⁻²×A²)</td>
</tr>
<tr>
<td>degree Celsius</td>
<td>°C</td>
<td>temperature relative to 273.15 K</td>
<td>K</td>
</tr>
<tr>
<td>lumen</td>
<td>lm</td>
<td>luminous flux</td>
<td>cd×sr (cd)</td>
</tr>
<tr>
<td>lux</td>
<td>lx</td>
<td>illuminance</td>
<td>lm/m² (m²×cd)</td>
</tr>
</tbody>
</table>

Table 2: Derived Units
Table 3 shows further units of physics derived from the base units.

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
<th>Derived Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>square meter</td>
<td>m²</td>
</tr>
<tr>
<td>volume</td>
<td>cubic meter</td>
<td>m³</td>
</tr>
<tr>
<td>speed, velocity</td>
<td>meter per second</td>
<td>m/s</td>
</tr>
<tr>
<td>acceleration</td>
<td>meter per second squared</td>
<td>m/s²</td>
</tr>
<tr>
<td>wave number</td>
<td>reciprocal meter</td>
<td>m⁻¹</td>
</tr>
<tr>
<td>mass density</td>
<td>kilogram per cubic meter</td>
<td>kg/m³</td>
</tr>
<tr>
<td>specific volume</td>
<td>cubic meter per kilogram</td>
<td>m³/kg</td>
</tr>
<tr>
<td>current density</td>
<td>ampere per square meter</td>
<td>A/m²</td>
</tr>
<tr>
<td>magnetic field strength</td>
<td>ampere per meter</td>
<td>A/m</td>
</tr>
<tr>
<td>amount-of-substance</td>
<td>mole per cubic meter</td>
<td>mol/m³</td>
</tr>
<tr>
<td>concentration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>luminance</td>
<td>candela per square meter</td>
<td>cd/m²</td>
</tr>
<tr>
<td>mass fraction</td>
<td>kilogram per kilogram</td>
<td>kg/kg = 1</td>
</tr>
</tbody>
</table>

Table 3: Further Units
Table 4 shows standard prefixes for SI units of measure.

<table>
<thead>
<tr>
<th>Prefix name</th>
<th>deca</th>
<th>hecto</th>
<th>kilo</th>
<th>mega</th>
<th>giga</th>
<th>tera</th>
<th>peta</th>
<th>exa</th>
<th>zetta</th>
<th>yotta</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix symbol</td>
<td>da</td>
<td>h</td>
<td>k</td>
<td>M</td>
<td>G</td>
<td>T</td>
<td>P</td>
<td>E</td>
<td>Z</td>
<td>Y</td>
</tr>
<tr>
<td>Factor</td>
<td>$10^1$</td>
<td>$10^2$</td>
<td>$10^3$</td>
<td>$10^6$</td>
<td>$10^9$</td>
<td>$10^{12}$</td>
<td>$10^{15}$</td>
<td>$10^{18}$</td>
<td>$10^{21}$</td>
<td>$10^{24}$</td>
</tr>
</tbody>
</table>

Table 4: Standard Prefixes for SI Units of Measure

Table 5 shows fractions of SI units of measure.

<table>
<thead>
<tr>
<th>Prefix name</th>
<th>deci</th>
<th>centi</th>
<th>milli</th>
<th>micro</th>
<th>nano</th>
<th>pico</th>
<th>femto</th>
<th>atto</th>
<th>zepto</th>
<th>yocto</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prefix symbol</td>
<td>d</td>
<td>c</td>
<td>m</td>
<td>µ</td>
<td>n</td>
<td>p</td>
<td>f</td>
<td>a</td>
<td>z</td>
<td>y</td>
</tr>
<tr>
<td>Factor</td>
<td>$10^0$</td>
<td>$10^{-1}$</td>
<td>$10^{-2}$</td>
<td>$10^{-3}$</td>
<td>$10^{-6}$</td>
<td>$10^{-9}$</td>
<td>$10^{-12}$</td>
<td>$10^{-15}$</td>
<td>$10^{-18}$</td>
<td>$10^{-21}$</td>
</tr>
</tbody>
</table>

Table 5: Fractions
These “pictures” are meant as an eye opener, a “teaser”.

\[
\begin{align*}
\rho &= mc^2 \\
E &= mc^2 \\
F_v &= \sum F_n \\
E &= \frac{1}{2} mc^2 \\
M &= F_d \cos \alpha \\
F_b &= S \eta p g \frac{2}{m} \\
\end{align*}
\]
And these formulas likewise!

\[
\begin{align*}
\text{Efficiency} &= \frac{W}{Q_h} \\
\frac{Q_c}{Q_h} &= \frac{T_c}{T_h} \\
\text{Efficiency} &= 1 - \left(\frac{Q_c}{Q_h}\right) = 1 - \left(\frac{T_c}{T_h}\right) \\
\text{Coefficient of performance} &= \frac{Q_h}{W} \\
\text{Coefficient of performance} &= \frac{1}{1 - \left(\frac{Q_c}{Q_h}\right)} = \frac{1}{1 - \left(\frac{T_c}{T_h}\right)} \\
p &= \frac{m}{V} \\
P &= \frac{F}{A} \\
\Delta P &= pgh \\
F_{\text{buoyancy}} &= W_{\text{water displaced}} \\
p_1A_1v_1 &= p_2A_2v_2 \\
P_1 + \frac{1}{2}pv_1^2 + pgy_1 &= P_2 + \frac{1}{2}pv_2^2 + pgy_2
\end{align*}
\]

- The point in bringing this material is
  - that when modelling, i.e., describing domains
  - we must be extremely careful in not falling into the trap
  - of modelling physics, etc., types as we do in programming!
3.2.2. What Are We to Learn from this Exposition?

- Physics units can be highly “structured”\(^{10}\).

- What Are We to Learn from this Exposition?

- I think it is this:
  - It is customary, in programs of languages from *Algol 60* via *Pascal* to *Java*,
    - to assign *float* or *double\(^{11}\)* types, as in *Java*,
    - to [constants or] variables
      - that for example represent values of physics.
  - *So rather completely different types of physics units are all cast into a same, simple-minded, “number” type.*
  - *No chance, really, for any meaningful type checking.*

---

\(^{10}\) For example, *Newton*: \(\text{kg} \times \text{m} \times \text{s}^{-2}\), *Volt* = \(\text{kg} \times \text{m}^2 \times \text{s}^{-3} \times \text{A}^{-1}\), etc.

\(^{11}\) representing single-, resp. double-precision 32-bit IEEE 754 floating point values
3.3. **Attribute Types, Scales and Values: Some Thoughts**

- This section further elaborates on the treatment of attributes given in Sect. 2.2.3, Slides 29–36.
- The elaboration is only sketched.
- It need be studied, in detail.
• The elaboration is this:

  ♦ The attr\_A observer function,  
    for a part \( p \) of sort \( P \),  
    such as defined in Sect. 2.2.3 (Slide 30)  
    yields values of type \( A \).

  ♦ In the revised understanding of attributes  
    the attr\_A observer is now to yield  
    both the type, \( AT \), and the value, \( AV \), of attribute \( A \):  

    \[
    \text{type} \\
    \hspace{1cm} AT, AV \\
    \text{value} \\
    \hspace{1cm} \text{attr}_A: P \rightarrow AT \times AV \\
    \]

  ♦ You may think of \( A \) being defined by \( AT \times AV \).
• The revision is further that a domain analysis & description of the operations over attributes values, \( \theta \):

\[
\theta : A_i \times A_j \times \ldots \times A_k \rightarrow V
\]

be carefully checked – such as hinted at in Sect. 3.1 (Slides 65–68).

• Whether such operator-checks be researched and documented
  - “once-and-for-all” for given “standard” domains, by domain scientists, or
  - per domain model, by domain engineers,
    in connection with specific software development projects

is left for you to decide!

• These operator-checks,
  - if not pursued, results in implicit semantics, and
  - if pursued, results in explicit semantics.
4. Conclusion

4.1. What Have We Achieved?

- We have suggested that the issue of implicit semantics [2] be resolved
  - by providing a carefully analysed and described domain model [1] prior to requirements capture and software design,
  - a both informally annotated and formally specified model that goes beyond [1] in its treatment of attributes
  - in that these are now endowed with types [and possibly scales (or fractions)] and that each specific domain model analyses and formalises the constraints that operations upon attribute values are carefully analysed, statically.
4.2. Domain Descriptions as Basis for Requirements Prescriptions

• This invited talk covers but one aspect of software development.


• [9] offers a systematic approach to requirements engineering based on domain descriptions. It is this approach that justifies our claim that domain modelling “alleviate the issue of implicit semantics.”

• [10] presents an operational/denotational semantics of the manifest domain analysis & description calculus of [1].

• [11]\textsuperscript{12} shows that to every manifest mereology there corresponds a CSP expression.

• [12] muses over issues of software simulators, demos, monitors and controllers.

\textsuperscript{12}Accepted for publication in Journal of Logical and Algebraic Methods in Programming, 2018.
4.3. **What Next?**

- Well, there is a lot of fascinating research to be done now.
- Studying analysis & description techniques for attribute types, values and constraints.
- And for engineering their support.

4.4. **Thanks**

- to J. Paul Gibson and Dominique Méry for inviting me,
- to J. Paul Gibson for organising my flights, hotel and registration, and
- to Dominique Méry for his patience in waiting for my written contribution.
5. Bibliographical Notes

5.1. References to Draft Domain Descriptions

- *Urban Planning* [13],  
- *Documents* [14],  
- *Credit Cards* [15],  
- *Weather Systems* [16],  
- *The Tokyo Stock Exchange* [17],  
- *Pipelines* [18],  
- *Road Transportation* [19],  
- *Trans.-based Web Software* [20],  
- “*The Market*” [21],  
- *Container Lines* [22] and  
- *Railways* [23, 24, 25, 26, 27].

- I apologise for the numerous references to own reports and publications.
5.2. References


