Abstract
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 wrote this paper 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
1 Introduction
   1.1 On the Issues of Implicit and Explicit Semantics
   1.2 A Triptych of Software Engineering
   1.4 Semantics
   1.5 Method & Methodology
   1.6 Computer & Computing Sciences
2 The Analysis & Description Prompts
   2.1 Endurants: Parts, Components and Materials
   2.2 Internal Qualities
      2.2.1 Unique Identifiers
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 it 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 paper we offer a calculus for analysing & describing domains (i.e., contexts), 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.

Proof of program, i.e., software code, correctness can be expressed as:

- \(D, S \models R\)

which we read as: proofs that software is correct with respect to requirements implies references to the domain.

### 1.3 Contexts \(^2\) \(\equiv\) Domains \(^1\)

Often the domain is referred to as the **context**. We treat contexts, i.e., domain descriptions as first class citizens \(^2\) \([2, Abstract, Page 1, lines 9–10]\). 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**\(^2\).

### 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.

### 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.**

---

\(^1\)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.

\(^2\)“The contexts … are treated as second-class citizens: in general, the modelling is implicit and usually distributed between the requirements model and the system model.” \([2, Abstract, Page 1, lines 9–12]\).
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, 12 pages, 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. See Fig. 1. The figure suggests a number of analysis and

---

\(^3\)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”.

---

Figure 1: An Ontology for Manifest Domains
description prompts. The domain analyser & describer is “positioned” at the top, the “root”. If what is observed can be conceived and described then it is an entity. If it can be described as a “complete thing” at no matter which given snapshot of time then it is 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 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. If the observed part, \(p:P\), is\(\text{composite}\) 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” observe\_part\_sorts to \(p\) yields an informal (i.e., a narrative) and a formal description:

**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.

Aircraft Example 2: Parts

1 An **aircraft** is composed from several parts of which we focus on

a a **position** part,

---

\(^4\) — ‘mereology’ will be explained next

\(^5\) Pragmatics is here used in the sense outlined in [4, Chapter 7, Pages 145–148].
b a travel dynamics part, and
c a display part.

type
1 AC, PP, TD, DP
value
1a obs_PP: AC \rightarrow PP
1b obs_TD: AC \rightarrow TD
1c obs_DP: AC \rightarrow DP

We have just summarised the analysis and description aspects of endurants in \textit{extension} (their “form”). We now summarise the analysis and description aspects of endurants in \textit{intension} (their “contents”). There are three kinds of intensional \textit{qualities} associated with parts, two with components, and one with materials. Parts and components, by definition, have \textit{unique identifiers}; parts have \textit{mereologies}, and all endurants have \textit{attributes}.

\section*{2.2 Internal Qualities}

\subsection*{2.2.1 Unique Identifiers}

Unique identifiers are further undefined tokens that uniquely identify parts and components. The description language observer \texttt{uid}\_\texttt{P}, when applied to parts \texttt{p}\_\texttt{P} yields the unique identifier, \texttt{π}:\texttt{Π}, of \texttt{p}. So the \texttt{observe}\_\texttt{part}\_\texttt{sorts}(\texttt{p}) invocation also yields the description text:

\begin{itemize}
  \item \texttt{...} [added to the narrative and]
  \item \texttt{type}
    \begin{itemize}
      \item \texttt{Φ \texttt{Π}_1, \texttt{Π}_2, ..., \texttt{Π}_m;}
    \end{itemize}
  \item \texttt{value}
    \begin{itemize}
      \item \texttt{Φ \texttt{uid}_\texttt{Π}_i : \texttt{P}_i \rightarrow \texttt{Π}_i},
    \end{itemize}
    repeated for all \texttt{m} part sorts \texttt{P}_i\texttt{s} and added to the formalisation.
\end{itemize}

\subsection*{Aircraft Example 3: Unique Identifiers}

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

\begin{itemize}
  \item \texttt{type}
    \begin{itemize}
      \item \texttt{PPI, TDI, DPI}
    \end{itemize}
  \item \texttt{value}
    \begin{itemize}
      \item \texttt{uid_PP: PP \rightarrow PPI}
      \item \texttt{uid_TD: TD \rightarrow TDI}
    \end{itemize}
\end{itemize}
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 \( \text{mereo}_P : P \rightarrow \mathcal{E}(\Pi_1, \ldots, \Pi_k) \) where \( \mathcal{E}(\Pi_1, \ldots, \Pi_k) \) is a type expression. So the \text{observe part sorts} \( (p) \) invocation also yields the description text:

\[
\begin{align*}
\text{value} & \quad \diamond \text{mereo}_P : P \rightarrow \mathcal{E}(\Pi_1, \ldots, \Pi_k) \\
\end{align*}
\]

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.

\[
\begin{align*}
\text{value} & \quad 3 \text{ mereo}_P : PP \rightarrow DPI \\
& \quad 4 \text{ mereo}_P : TP \rightarrow DPI \\
& \quad 4 \text{ mereo}_P : DP \rightarrow PPI \times TDI \\
\end{align*}
\]

2.2.3 Attributes

Attributes are the remaining qualities of endurants. The analysis prompt \text{obs attributes} applied to an endurant yields a set of type names, \( A_1, A_2, \ldots, A_t \), of attributes. They imply the additional description text:

\[
\begin{align*}
\text{value} & \quad \diamond A_1, A_2, \ldots, A_t \\
\end{align*}
\]
value

$\forall$ attr$_{A_i} \colon E \rightarrow A_i$

repeated for all $t$ attribute sorts $A_i$s!

---

**Aircraft Example 5:** Position Attributes

6 Position parts have longitude, latitude and altitude attributes.

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO, LA, AL</td>
<td>attr_LO: PP $\rightarrow$ LO</td>
</tr>
<tr>
<td></td>
<td>attr_LA: PP $\rightarrow$ LA</td>
</tr>
<tr>
<td></td>
<td>attr_AL: PP $\rightarrow$ AL</td>
</tr>
</tbody>
</table>

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

7 Travel dynamics parts have velocity$^6$ and acceleration$^7$.

<table>
<thead>
<tr>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VEL, ACC</td>
<td>attr_VEL: TD $\rightarrow$ VEL</td>
</tr>
<tr>
<td></td>
<td>attr_ACC: TD $\rightarrow$ ACC</td>
</tr>
</tbody>
</table>

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.

---

**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 recordings,

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

<table>
<thead>
<tr>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LO, LA, AL, VEL, ACC</td>
</tr>
</tbody>
</table>

---

$^6$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.
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.

type
11 dLO, dLA, dAL
11 dVEL, dACC
value
11 attr_dLO: DP → dLO
11 attr_dLA: DP → dLA
11 attr_dAL: DP → dAL
11 attr_dVEL: DP → dVEL

11 attr_dACC: DP → dACC
11 r2dLO, d2rLO: rLO ↔ dLO
11 r2dLA, d2rLA: rLA ↔ dLA
11 r2dAL, d2rAL: rAL ↔ dAL
11 r2dVEL, d2rVEL: rVEL ↔ dVEL
11 r2dACC, d2rACC: rACC ↔ dACC

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.

\[
\text{Aircraft Example 10: An Axiom}
\]

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

axiom
14 \[ \forall ac:AC \bullet \]
14 let \( (pp,td,di) = (\text{obs\_PP}(ac),\text{obs\_TD}(ac),\text{obs\_DP}(ac)) \) in
14 let \( (lo,la,at) = (\text{attr\_LO}(pp),\text{attr\_LA}(pp),\text{attr\_AT}(pp)) \),
14 \( (vel,acc,dir) = (\text{attr\_VEL}(td),\text{obs\_ACC}(td)) \),
14 \( (dlo,dla,dat) = (\text{attr\_dLO}(di),\text{attr\_dLA}(di),\text{attr\_dAT}(di)) \),
14 \( (dvel,dacc) = (\text{attr\_dVEL}(di),\text{obs\_dACC}(di)) \) in
14 \( (dlo,dla,dat) = (r2dLO(a2rLO(lo)),r2dLA(a2rLA(la)),r2dAL(a2rAL(at))) \)
14 \( \wedge (dvel,dacc) = (r2dVEL(a2rVEL(vel)),r2dACC(a2rACC(acc))) \)
14 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.

\[
\text{Aircraft Example 11: An Informal Story}
\]

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, ch ! m, and receipt, ch ? m, of messages, m:M, over channels, ch.

type M
channel ch:M

Aircraft Example 12: Channels

For this example we focus only on communications from the position and travel dynamics behaviours to the display behaviour.

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 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.
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:

\[ \text{behaviour: } ... \rightarrow ... \text{ Unit} \]

That a process takes no argument is ”revealed” by a “leading” Unit:

\[ \text{behaviour: } \text{ Unit} \rightarrow ... \]

That a process accepts channel, viz.: ch, inputs, including accesses an external attribute A, is “revealed” in the function signature as follows:

\[ \text{behaviour: } ... \rightarrow \text{ in ch ... , resp. in attr}_{A,ch} \]

That a process offers channel, viz.: ch, outputs is “revealed” in the function signature as follows:

\[ \text{behaviour: } ... \rightarrow \text{ out ch ...} \]

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

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

where ARG can be any type expression:

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

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., (...). Static attributes become behaviour definition (body) constant values. Inert, reactive and autonomous attributes become references
to channels, say \texttt{ch\_dyn}, such that when an inert, reactive and autonomous attribute value is required it is expressed as \texttt{ch\_dyn ?}. Programmable and biddable attributes become arguments which are passed on to the tail-recursive invocations of the behaviour, and possibly updated as specified in the body of the definition of the behaviour, i.e., (...).

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:

\[
\text{beh}_{\pi:\Pi}: \text{me:MT} \times \text{sa:SA} \rightarrow \text{ca:CA} \rightarrow \text{ichns}(ea:EA) \ \text{in\_out\ ichs}(\text{me}) \ \text{Unit}
\]

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) \(\text{ichs}(\text{me})\) 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

We omit the signature of the aircraft behaviour.

20 The signature of the \textit{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 \textit{display} behaviour.

21 The signature of the \textit{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 \textit{display} behaviour.

22 The signature of the \textit{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 \textit{position}, respectively the \textit{travel dynamics} behaviours.

**Aircraft Example 14:** Part Behaviour Signatures, I/II

\begin{verbatim}
22 type DA = (dLA\times dLO\times dAL)\times(dVEL\times dACC)
22 value
20 position: PI \times DPI \rightarrow 
20 in attr\_LO\_ch, attr\_LA\_ch, attr\_AL\_ch, out po\_di\_ch Unit
21 travel\_dynamics: TDI \times DPI \rightarrow 
21 in attr\_VEL\_ch, attr\_ACC\_ch, attr\_DIR\_ch, out td\_di\_ch Unit
22 display: DI \times (PPI\times TDI) \rightarrow DA \rightarrow in po\_di\_ch, td\_di\_ch Unit
\end{verbatim}
2.4.6 Behaviour Compilations

**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, \( M_{cP_{uid}}(p) \), 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 \( p_2: P_2 \), ..., and \( p_n \) is compiled from \( p_n: P_n \). The domain description “compilation” schematic below “formalises” the above.

**Process Schema: Abstract**

\[
\text{is composite}(p)
\]

\[
\text{compile}_\text{process}: P \rightarrow \text{RSL-Text}
\]

\[
\text{compile}_\text{process}(p) \equiv \begin{align*}
M_{cP_{uid}}(p) & (\text{obs.mereo}_P(p), S_A(p))(C_A(p)) \\
& \parallel \text{compile}_\text{process}(\text{obs.part}_{P_1}(p)) \\
& \parallel \text{compile}_\text{process}(\text{obs.part}_{P_2}(p)) \\
& \quad \ldots \\
& \parallel \text{compile}_\text{process}(\text{obs.part}_{P_n}(p))
\end{align*}
\]

The text macros: \( S_A \) and \( C_A \) were informally explained above. Part sorts \( P_1, P_2, \ldots, P_n \) are obtained from the \textit{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**

\[
\text{compile}(ac) \equiv \begin{align*}
23 & \quad \text{compile}(\text{obs.PP}(p)) \\
23a & \quad \text{compile}(\text{obs.TD}(p)) \\
23b & \quad \text{compile}(\text{obs.DI}(p))
\end{align*}
\]

**Atomic Behaviours**
**Process Schema: is_atomic(p)**

\[
\text{compile}_\text{process}: P \rightarrow \text{RSL-Text} \\
\text{compile}_\text{process}(p) \equiv \\
\mathcal{M}_{\text{uid}_p}^{\text{P}(p)}(\text{obs}_\text{mereo}_p, \text{S}_A(p))(\text{C}_A(p))
\]

---

**Aircraft Example 16: Atomic Behaviours**

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

value

23a compile(\text{obs}_\text{PP}(p)) \equiv \\
23a \quad \text{position}(\text{uid}_\text{PP}(p), \text{mereo}_\text{PP}(p)) \\
23b compile(\text{obs}_\text{TD}(p)) \equiv \\
23b \quad \text{travel}_\text{dynamics}(\text{uid}_\text{TD}(p), \text{mereo}_\text{TD}(p)) \\
25 \quad \text{init}_\text{DA} = \ldots \\
23c compile(\text{obs}_\text{DI}(p)) \equiv \\
23c \quad \text{display}(\text{uid}_\text{DI}(p), \text{mereo}_\text{DI}(p))(\text{init}_\text{DA})

In the above we have already subsumed the *atomic behaviour definitions*, see next, and directly inserted the \( F \) definitions.

---

### 2.4.7 Atomic Behaviour Definitions

**Process Schema IV: Atomic Core Processes**

value

\[
\mathcal{M}_{\pi,\Pi}: \text{me}: \text{MT} \times \text{sa}: \text{SA} \rightarrow \text{ca}: \text{CA} \rightarrow \\
\text{in ichns}(\text{ea}: \text{EA}) \text{ in, out iochs}(\text{me}) \text{ Unit} \\
\mathcal{M}_{\pi,\Pi}(\text{me}, \text{sa})(\text{ca}) \equiv \\
\quad \text{let } (\text{me}', \text{ca}') = F_{\pi,\Pi}(\text{me}, \text{sa})(\text{ca}) \text{ in } \\
\quad \mathcal{M}_{\pi,\Pi}(\text{me}', \text{sa})(\text{ca}') \text{ end} \\
F_{\pi,\Pi}: \text{me}: \text{MT} \times \text{sa}: \text{SA} \rightarrow \text{CA} \rightarrow \\
\text{in ichns}(\text{ea}: \text{EA}) \text{ in, out iochs}(\text{me}) \rightarrow \text{MT} \times \text{CA}
\]

---

**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(pπ,dπ) ≡
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(pπ,dπ) 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.

value
21 travel_dynamics(tdπ,dπ) ≡
29 let (vel,acc) = (attr_VEL_ch?, attr_ACC_ch?) in
30 td_di_ch ! (a2rVEL(vel), a2rACC(acc)) ;
31 travel_dynamics(tdπ,dπ) end

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 conversion functions are extensions of the ones introduced earlier.

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

34 type 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.

3 Calculations in Classical Domains: Some Simple Observations

This section covers three loosely related topics: Sect. 3.1 muses over properties of some attribute values. Then, Sect. 3.2 we recall some facts about types, scales and values of measurable units in physics. The previous leads us, in Sect. 3.3 to consider further detailing the concept of attributes such as we have covered it in Sect. 2.2.3, Pages 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, ≤, 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 ⊕ 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:

\[
\begin{array}{ll}
type & value \\
T = \text{Month} \times \text{Day} \times \text{Year} \times \text{Hour} \times \text{Minute} \times \text{Sec}... & <,\leq,=,\geq,>: T \times T \rightarrow \text{Boole} \\
\text{Tl} = \text{Days} \times \text{Hours} \times \text{Minutes} \times \text{Seconds} \times... & -: T \times T \rightarrow \text{Tl} \ 	ext{pre} \ t \rightarrow t': t' \leq t \\
\text{Month} = \{1,2,3,4,5,6,7,8,9,10,11,12\} & <,\leq,=,\geq,>: \text{Tl} \times \text{Tl} \rightarrow \text{Bool} \\
\text{Day} = \{1,2,3,4,...,28,29,30,31\} & -,: \text{Tl} \times \text{Tl} \rightarrow \text{Tl} \\
\text{Hour,Hours} = \{0,1,2,3,...,21,22,23\} & \ast: \text{Tl} \times \text{Real} \rightarrow \text{Tl} \\
\text{Minute,Minutes} = \{0,1,2,3,...,56,57,58,59\} & /: \text{Tl} \times \text{Tl} \rightarrow \text{Real} \\
\text{Second,Seconds} = \{0,1,2,3,...,56,57,58,59\} & ... \\
\text{Days} = \text{Nat} \\
\end{array}
\]

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

---

\(^9\)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!
probably be well advised to have these represent regular sampling, or at least time-stamped. One can also define rate of change of temperature.

type
  Temp, MeanTemp, Degrees, TempIntv = Degrees
value
  mean: Temp-set × Nat → MeanTemp
  -: Temp × Temp → TempIntv

type
  TST = (Temp × T)-set
value
  avg: TST → MeanTemp

type
  TimeUnit = \{"year", "month", "day", "hour", ...\}
  RoTC = TempIntv × TimeUnit

Etcetera. We leave it to the reader to speculate on which operations one can perform on a persons’ 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 temp.</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 on the facing page shows the units of physics derived from the base units.
Table 3 on page 20 shows further units of physics derived from the base units.
Table 4 on page 20 shows standard prefixes for SI units of measure.
Table 5 on page 21 shows fractions of SI units of measure.
These “pictures” are meant as an eye opener, a “teaser”.

© D.Bjørner 2017, Fredsvej 11, 2840 Holtes, Denmark. November 16, 2017: 00:23
<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 m/m</td>
<td></td>
</tr>
<tr>
<td>steradian</td>
<td>sr</td>
<td>solid angle m²/m²</td>
<td></td>
</tr>
<tr>
<td>Hertz</td>
<td>Hz</td>
<td>frequency s⁻¹</td>
<td></td>
</tr>
<tr>
<td>newton</td>
<td>N</td>
<td>force, weight kg×m×s⁻²</td>
<td></td>
</tr>
<tr>
<td>pascal</td>
<td>Pa</td>
<td>pressure, stress N/m²</td>
<td></td>
</tr>
<tr>
<td>joule</td>
<td>J</td>
<td>energy, work, heat N×m</td>
<td></td>
</tr>
<tr>
<td>watt</td>
<td>W</td>
<td>power, radiant flux J/s</td>
<td></td>
</tr>
<tr>
<td>coulomb</td>
<td>C</td>
<td>electric charge s×A</td>
<td></td>
</tr>
<tr>
<td>volt</td>
<td>V</td>
<td>voltage, electromotive force W/A (kg×m²×s⁻³×A⁻¹)</td>
<td></td>
</tr>
<tr>
<td>farad</td>
<td>F</td>
<td>capacitance C/V (kg⁻¹×m⁻²×s⁴×A²)</td>
<td></td>
</tr>
<tr>
<td>ohm</td>
<td>Ω</td>
<td>electrical resistance V/A (kg×m²×s³×A²)</td>
<td></td>
</tr>
<tr>
<td>siemens</td>
<td>S</td>
<td>electrical conductance A/V (kg×m²×s³×A²)</td>
<td></td>
</tr>
<tr>
<td>weber</td>
<td>Wb</td>
<td>magnetic flux V×s (kg×m²×s⁻²×A⁻¹)</td>
<td></td>
</tr>
<tr>
<td>tesla</td>
<td>T</td>
<td>magnetic flux density Wb/m² (kg×m²×A⁻¹)</td>
<td></td>
</tr>
<tr>
<td>henry</td>
<td>H</td>
<td>inductance Wb/A (kg×m²×s⁻²×A²)</td>
<td></td>
</tr>
<tr>
<td>degree Celsius</td>
<td>°C</td>
<td>temperature relative to 273.15 K K</td>
<td></td>
</tr>
<tr>
<td>lumen</td>
<td>lm</td>
<td>luminous flux cd×sr (cd)</td>
<td></td>
</tr>
<tr>
<td>lux</td>
<td>lx</td>
<td>illuminance lm/m² (m²×cd)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Derived Units

And these formulas likewise!

\[
\text{Efficiency} = \frac{W}{Q_h} \\
Q_{in} = \frac{Q_{in}}{Q_h} = \frac{T_C}{T_h} \\
\text{Efficiency} = 1 - \left(\frac{Q_{in}}{Q_{in}}\right) = 1 - \left(\frac{T_C}{T_h}\right) \\
\text{Coefficient of performance} = \frac{Q_h}{W} \\
\text{Coefficient of performance} = 1 - \frac{1}{Q_{in}} = \frac{1}{1 - \frac{T_C}{T_h}} \]

\[
\rho = \frac{m}{V} \\
\rho = \frac{F}{A} \\
\Delta P = \rho g h \\
F_{\text{hydrostatic}} = W_{\text{water displaced}} \\
\rho_1 A_1 v_1 = \rho_2 A_2 v_2 \\
\]

Carnot Engine Bernoulli Flow

The point in bringing this material is that when modelling, i.e., describing domains we...
<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 concentration</td>
<td>mole per cubic meter</td>
<td>mol/m³</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>
<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>10⁰</td>
<td>10¹</td>
<td>10²</td>
<td>10³</td>
<td>10⁶</td>
<td>10⁹</td>
<td>10¹²</td>
<td>10¹⁵</td>
<td>10¹⁸</td>
<td>10²¹</td>
</tr>
</tbody>
</table>

Table 4: Standard Prefixes for SI Units of Measure

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?

We see from the previous section, Sect. 3.2, that 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 double¹¹ 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.

3.3 Attribute Types, Scales and Values: Some Thoughts

This section further elaborates on the treatment of attributes given in Sect. 2.2.3, Pages 7–9. 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 (Page 8) 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:

\[
\text{type} \quad AT, AV \\
\text{value} \quad \text{attr}_A: P \to AT \times AV
\]

¹⁰For example, Newton: kg×m×s⁻², Volt = kg×m²×s⁻³×A⁻¹, etc.
¹¹representing single-, resp. double-precision 32-bit IEEE 754 floating point values
<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

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 on page 17.

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 paper 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] shows that to every manifest mereology there corresponds a CSP expression.
- [12] muses over issues of software simulators, demos, monitors and controllers.

---

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
  - Road Transportation

- Documents
  - Transaction-based Web Software

- Credit Cards
  - “The Market”

- Weather Information Systems

- The Tokyo Stock Exchange

- Pipelines
  - Container [Shipping] Lines

- Railway Systems

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

5.2 References


