Domain Science & Engineering
A Review of 10 Years Work
Invited Talk at the ZCC Fest, 20 October 2017, Changsha, China

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

October 20, 2017: 01:12 am
1. Introduction

- I survey recent work in the area of domain science & engineering.
- It is based on the triptych dogma:
  - before software can be designed we must understand its requirements,
  - and before we can prescribe the requirements we must understand the domain.
- Not the “whole world”, but “more than sufficient!”.

---

1 In our research into domain science & engineering we insist on first modelling the domain. Requirements engineering may then reveal whether we have described all the pertinent properties, or ...
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.
- 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.
1.1. Recent Papers

Over the last decade I have iterated a number of investigations of aspects of the *triptych* dogma. This has resulted in a number of papers (and revised reports):

- Manifest Domains: Analysis & Description [1]  
  *FAoC, March 2017*
- Domain Facets: Analysis & Description [2, 3]
- Formal Models of Processes and Prompts [4, 5]
- To Every Manifest Mereology a CSP Expression [6]  
  *LAMP, early 2018*
- From Domain Descriptions to Requirements Prescriptions [7, 8]
1.2. Recent Experiments

Applications of the domain science and engineering outlined in [1]–[9] are exemplified in reports and papers on experimental domain analysis & description. Examples are:

- Urban Planning [10],
- Documents [11],
- Credit Cards [12],
- Weather Systems [13],
- The Tokyo Stock Exchange [14],
- Pipelines [15],
- Road Transportation [16],
- Web–based Software [17],
- “The Market” [18],
- Container Lines [19] and
- Railways [20, 21, 22, 23, 24].
1.3. My Emphasis on Software Systems

• An emphasis in my work has been on
  ◦ research into and experiments with application areas
  ◦ that required seemingly large scale software.
  ◦ Not on tiny, beautiful, essential data structures and algorithms.

• I first worked on the proper application of formal methods in software engineering

• at the IBM Vienna Laboratory in the early 1970s.

• That was to the formalisation of the semantics of IBM’s leading programming language then, PL/I,

• and to a systematic development of a compiler for that language.

• The latter never transpired.
• Instead I got the chance to formulate the stages of development of a compiler from a denotational semantics description to so-called “running code” [25, 1977].

• That led, from 1978 onwards, to two MSc students and a colleague and I working on a formal description of the CCITT Communications High Level Language, CHILL and its compiler [26, 27].

• And that led, in 1980, to five MSc students of ours producing a formal description of a semantics for the US DoD Ada programming language [28].

• And that led to the formation of Dansk Datamatik Center [29] which embarked on the CHILL and Ada compiler developments [30, 31].
1. Introduction
1.3. My Emphasis on Software Systems

- **To my knowledge that project**
  - which was on time, at budget, and
  - with a history of less than 3% cost of original budget
  - for subsequent error correction
    over the first 20 years of use of that compiler
  - was a first, large, successful example
  - of the systematic use of formal methods
  - in large scale (42 man years) software development.
1.4. How Did We Get to Domain Science & Engineering?

• So that is how we came
  - from the semantics of programming languages
  - to the semantics of human-centered, manifest application domain software development.

• Programming language semantics
  - has to do with the meaning of abstract concepts
  - such as programs, procedures, expressions, statements, GOTOs, labels, etc.
• Domain semantics, for manifest domains,
  • in so far as we can narrate and formalize it, or them,
  • must capture some “meanings”
    of the manifest objects that we can touch and see,
  • of the actions we perform on them,
  • and of the sentences by means of which
    we talk about those phenomena in the domain.
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 contributions over the years have contributed**
  
  ◦ to methods for software design
  ◦ and, now, for the last many years,
    methods for domain analysis & description.
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 Papers

• IM²HO I consider the first of the papers reviewed, [1], my most important paper.

  • It was conceived of last\(^2\),
  • after publication of three of the other papers [3, 8, 9].
  • Experimental evidence then necessitated extensive revisions to these other papers, resulting in [2, 7, 34].

• In the following I will review [1].

• I will then – one to two slides – briefly summarize

  • [2] and [7] (they are methodology-, cum domain engineering-, oriented), and
  • [4, 6] (which are domain science-oriented).

\(^2\)Publication [32, 33] is a predecessor of [1].
3. **Manifest Domains: Analysis & Description** [1]

- This work grew out of many years of search
  - for principles, techniques and tools for
  - systematically analyzing and describing manifest domains.

- By a manifest domain we shall understand a domain whose entities we can observe and whose endurants we can touch!
3.1. **A Domain Ontology**

3.1.1. **Parts, Components and Materials**

- The result became a calculus of analysis and description prompts\(^3\).
  - 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*. 

An Ontology for Manifest Domains

"a thing"

Describable is_entity

Non-describable

is_endurant

is_perdurant

Action Event Behaviour Channel

Signature Definition

is_material

obs_material

endurants: form

obs_part_type

has_concrete_type

obs_concrete_type

has_materials

obs_materials

has_components

obs_components

is_atomic

is_discrete

is_continuous

is_permutable

is_composite

is_part

is_component

obs_comp_sorts

obs_part_sorts

obs_attributes

obs_uid

obs_mereology

attributes

unique identifiers

static

dynamic

inert

reactive

active

autonomous

biddable

programmable

imply

attribute categories

analysis prompts

description prompts

obs_ is an abbreviation for 'observe'
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 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**.
• 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 as **parts**, 
  - else we shall call them **components**.

• The continuous endurants we shall also refer to as *(gaseous or liquid)* **materials**.

[^4]: ‘mereology’ will be explained next
Materials have types (i.e., are of sorts): $M_i$.

- Observing the (one) material, of type $M$, of an endurant $e$ of sort $E$
- is expressed as `obs_material(e)`
- which yields some narrative and some formal *description text*:

  **Narrative:**
  
  - ...

  **Formal:**
  
  - type
    
    `* M`
  
  - value
    
    `* obs_M: E → M`

- The narrative text (...) narrates what the formal text expresses$^5$.

---

$^5$ not how it expresses it, as, here, in the RAISE [35] Specification Language, RSL [36].
• Parts are either **atomic** or **composite** and all parts have
  ◦ *unique identifiers*,
  ◦ *mereology* and
  ◦ *attributes*.

• Atomic parts *may have* one or more **materials**
  ◦ in which case we may observe these materials: \texttt{obs\_materials}(p)
  ◦ which yields the informal and formal description:
• Narrative:
  ◦ ...

• Formal:
  ◦ type
    ◦ $M_1, M_2, ..., M_n$
  ◦ value
    ◦ $\text{obs}_{M_i}: P \rightarrow M_i$

repeated for all $n$ $M_i$s!
• If the observed part, \( p:P \), is composite
  
  \( \diamond \) then we can observe the part sorts, \( P_1, P_2, \ldots, P_m \) of \( p \):
  
  \( \diamond \) observe_part_sorts\((p) \) which yields the informal and formal description:

  
  • **Narrative:**
    
    \( \diamond \ldots \)

  • **Formal:**
    
    \( \diamond \) type
      
      \( \diamond P_1, P_2, \ldots, P_m, \)

    \( \diamond \) value
      
      \( \diamond \) obs\(_{P_i}\) : \( P \rightarrow P_i, \)

    repeated for all \( m \) part sorts \( P_i \)'s”!
• Part sorts may have a concrete type: $\text{has\_concrete\_type}(p)$

• in which case $\text{observe\_concrete\_part\_type}(p)$ yields

<table>
<thead>
<tr>
<th>Narrative:</th>
</tr>
</thead>
<tbody>
<tr>
<td>⋄ ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>⋄ type:</td>
</tr>
<tr>
<td>◦ $T = P − \text{set}$,</td>
</tr>
<tr>
<td>⋄ value</td>
</tr>
<tr>
<td>◦ $\text{obs}_T: P \rightarrow K−\text{set}$</td>
</tr>
</tbody>
</table>

where $K−\text{set}$ is one of the concrete type forms, and where $K$ is some sort.
• **Components**, i.e., discrete endurants for whom we do not consider possible mereologies,

  ◊ can be observed from materials, $m : M$,
  or are just observed of discrete endurants, $e : E$:

  ◊ **obs_comp_sorts**\((em)\) which yields

  the informal and formal description:

<table>
<thead>
<tr>
<th>• Narrative:</th>
</tr>
</thead>
<tbody>
<tr>
<td>◊ ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>• Formal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>◊ <strong>type:</strong></td>
</tr>
<tr>
<td>◊ $C_1, C_2, ..., C_n$</td>
</tr>
<tr>
<td>◊ <strong>value</strong></td>
</tr>
<tr>
<td>◊ <strong>obs(_{C_i})</strong>: $(E</td>
</tr>
</tbody>
</table>

  repeated for all $n$ component sorts $C$s” to the formal text!
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*. 
3.1.2. Unique identifiers

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

- ... [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.
3.1.3. 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 **mereo_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:

  - ... [added to the narrative and]
  - value
    - **mereo_P_i**: \( P_i \rightarrow \mathcal{E}_i(\Pi_{i,j}, ..., \Pi_{i,k}) \) [added to the formalisation]
• Example:

◊ The mereologies, \((i, o)\), of pipe units in a pipeline system
◊ thus express, for each kind of pipe unit,

whether it is

◊ a well,
◊ a linear pipe,
◊ a fork,
◊ a join,

◊ the identities of the zero, one or two pipe units that it is
“connected” to on the input, \(i\), respectively the output, \(o\), side:

◊ for well \((0, 1)\),
◊ for pipe \((1, 1)\),
◊ for fork \((1, 2)\),
◊ for join \((2, 1)\),

◊ for valve \((1, 1)\),
◊ for pump \((1, 1)\),
◊ for sink \((1, 0)\)
3.1.4. Attributes

- Attributes are the remaining qualities of endurants.
  - The analysis prompt \texttt{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:

<table>
<thead>
<tr>
<th>Narrative:</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Formal:</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
</tr>
<tr>
<td>( A_1, A_2, ..., A_t )</td>
</tr>
<tr>
<td>value</td>
</tr>
<tr>
<td>( \text{attr}_{A_i}: E \rightarrow A_i )</td>
</tr>
</tbody>
</table>

repeated for all \( t \) attribute sorts \( A_i \)'s!
**Examples:**

Typical attributes of a person are

- Gender,
- Weight,
- Height,
- Birth date, etcetera.

Dynamic and static attributes of a pipe unit include

- current flow into the unit, per input, if any,
- current flow out of the unit, per output, if any
- current leak from the unit,
- guaranteed maximum flow into the unit,
- guaranteed maximum flow out of the unit,
- guaranteed maximum leak from the unit, etcetera.
Michael A. Jackson [37] categorizes 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.
3.2. From Manifest Parts to Domain Behaviours

- [1] then presents an interpretation, \( \tau \), which to manifest parts associate behaviours.

- These are then specified as CSP [38] processes.
3.2.1. The Idea — by means of an example

The term *train* can have the following “meanings”:

- The *train*, as an *endurant*,
  parked at the railway station platform,
  i.e., as a *composite part*.
- The *train*, as a *perdurant*,
  as it “speeds” down the railway track,
  i.e., as a *behaviour*.
- The *train*, as an *attribute*,

3.2.2. Atomic Parts

- Atomic parts translates into their core behaviours:

  - \( b_{\text{atom core}} \).

- The \textit{core} behaviours are tail recursively defined, that is, are cyclic.

  - \( b_{\text{atom core}}(...) \equiv (\ldots ; b_{\text{atom core}}(...)) \)

  - where \((...)\) indicate behaviour (i.e., function) arguments.
3.2.3. Composite Parts

- A composite part, $p$, “translates”, $\tau$, into the parallel composition of a core behaviour:
  - $b_{\text{core}}^{p_{\text{comp}}}(...)$, for part $p$,
  - with the parallel composition of the translations, $\tau$, for each of the parts, $p_1, p_2, ..., p_m$, of $p$, $(\tau(p_1) \parallel \tau(p_2) \parallel ... \parallel \tau(p_m))$
  - that is:
    - $\tau(p) \equiv b_{\text{core}}^{p_{\text{comp}}}(...) \parallel (\tau(p_1) \parallel \tau(p_2) \parallel ... \parallel \tau(p_m))$
3.2.4. **Concrete Parts**

- The translation of concrete part set, $t$, types, $t : T = K - \text{set}$, is

\[
\tau(t) \equiv \parallel \{ \tau(k_i) | k_i : K \cdot k_i \in t \}.
\]
3.2.5. 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 `ch_dyn`, such that when an inert, reactive and autonomous attribute value is required it is expressed as `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 [with]in the body of the definition of the behaviour, i.e., (...).
3.3. **Contributions of [1] – and Open Problems**

- For the first time we have, now, the beginnings of a calculus for developing domain descriptions.

  - In [32, 33] we speculate on laws that these analysis & description prompts (i.e., their “meanings”) must satisfy.
  - With this calculus we can now systematically develop domain descriptions [10–24].
  - I am right now working on understanding issues of *implicit/explicit* semantics\(^6\)

---

4. “The Other Papers”

4.1. Domain Facets: Analysis & Description [2, 3]

4.1.1. Overview

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

- [2] is an extensive revision of [3].
Both papers identify the following facets:

- **intrinsics**,  
- **support technologies**,  
- **rules & regulations**,  
- **scripts**,  
- **license languages**,  
- **management & organisation**, and  
- **human behaviour**.
• Recently I have “discovered” what might be classified as a domain facet: classes of *attribute semantics*:
  - the diversity of attribute semantics
  - resolving the issue of so-called implicit and explicit semantics.
  - I shall not cover this issue in this talk.
4.1.2. Contributions of [2, 3] – and Open Problems

- [2] now covers techniques and tools
  - for analyzing domains into these facets
  - and for their modeling.
- The issue of *license languages* are particularly intriguing.
- The delineations between the listed facets
  - is necessarily not as precise as one would wish:
  - we are dealing with an imprecise world,
    that of (manifest) domains.

---

*We have omitted a facet: *license languages.*
4.2. From Domain Descriptions to Requirements Prescriptions [7, 8]

4.2.1. Overview

• [7] outlines a calculus of refinements and extensions which applied to domain descriptions yield requirements prescriptions.
  ❖ As for [1] the calculus is to be deployed by human users, i.e., requirements engineers.
  ❖ Requirements are for a machine, that is, the hardware and software to be developed from the requirements.
  ❖ A distinction is made between domain, interface and machine requirements.

• I shall briefly cover these in another order.
4.2.1.1 Machine requirements

- **Machine requirements** are such which can be expressed using only technical terms of the machine:

  - performance and dependability
    - accessibility,
    - availability,
    - integrity,
    - reliability,

  and

  - development requirements
    - development process,
    - maintenance,
    - platform,

  - Within **maintenance requirements** there are
    - adaptive,
    - corrective,
    - perfective,

  - Within **platform requirements** there are
    - development,
    - execution,
    - maintenance,

  - Etcetera.

- [7] does not cover these. See instead [39, Sect. 19.6].
4.2.1.2 Domain Requirements

- *Domain requirements* are such which can be expressed using only technical terms of the domain.

- The are the following domain-to-requirements specification transformations:
  - projection,
  - instantiation,
  - determination,
  - extension and
  - fitting.

- I consider my work on these domain requirements issues the most interesting.
4.2.1.3 Interface Requirements

- *Interface requirements* are such which can be expressed only by using technical terms of both the domain and the machine.

- Thus interface requirements are about that which is *shared* between the domain and the machine:
  - *endurants* that are represented in machine storage as well as co-existing in the domain;
  - *actions* and *behaviours* that are performed while interacting with phenomena in the domain;

etc.
4.2.2. Contributions of [7, 8]

- [7] does not follow the “standard division” of requirements engineering into systems and user requirements etcetera.
  
  Instead [7] builds on domain descriptions and eventually gives a rather different “division of requirements engineering labour” – manifested in

  - the domain,
  - the interface and
  - the machine requirements paradigms,

  and these further into sub-paradigms, to wit:

  - projection,
  - instantiation,
  - determination,
  - extension and
  - fitting.
4.3. **Formal Models of Processes and Prompts [4, 5]**

4.3.1. **Overview**

- [1] outlines a calculus of prompts, to be deployed by human users, i.e., the domain analyzers & describers.

  - That calculus builds on the **assumption** that the domain engineers build, *in their mind*, i.e., conceptually,
  - a **syntactical** structure of the domain description,
  - although, what the domain engineers can “see & touch”
  - are **semantical** objects.
A formal model

- of the analysis and description prompt process
- and of the meanings of the prompts
- therefore is split into
  - a model for the process and
  - a model of the syntactic and semantics structures.
4.3.2. Contributions of [4]

- The contributions of [4] are
  - to suggest and carry through a “formalisation” of the *conceptual, syntactical* and *semantical* structures *perceived* by the domain engineer,
  - to formalise the *meaning of the informal* analysis & description prompts, and
  - to formalise the possible sets of sequences of valid prompts.
4.4. To Every Manifest Domain Mereology a CSP Expression [6]

4.4.1. Overview

• In [1] we have shown how parts can be endowed with mereologies.

   Mereology, as was mentioned earlier, is the study and knowledge of "parthood": of how parts are related
   - parts to parts, and
   - parts to "a whole".

   Mereology, as treated by us, originated with the Polish mathematician/logician/philosopher Stanislaw Leśniewski.
4.4.1.1 An Axiom System for Mereology

part_of: \( P : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \)
proper_part_of: \( PP : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \)
overlap: \( O : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \)
underlap: \( U : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \)
over_crossing: \( OX : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \)
under_crossing: \( UX : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \)
proper_overlap: \( PO : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \)
proper_underlap: \( PU : \mathcal{P} \times \mathcal{P} \rightarrow \text{Bool} \)
• Let $\mathcal{P}$ denote **part-hood**; $p_x$ is part of $p_y$, is then expressed as $\mathcal{P}(p_x, p_y)$.\(^8\)

  (1) Part $p_x$ is part of itself (reflexivity).
  (2) If a part $p_x$ is part of $p_y$ and, vice versa, part $p_y$ is part of $p_x$, then $p_x = p_y$ (anti-symmetry).
  (3) If a part $p_x$ is part of $p_y$ and part $p_y$ is part of $p_z$, then $p_x$ is part of $p_z$ (transitivity).

\[
\forall p_x : \mathcal{P} \bullet \mathcal{P}(p_x, p_x) \tag{1}
\]
\[
\forall p_x, p_y : \mathcal{P} \bullet (\mathcal{P}(p_x, p_y) \land \mathcal{P}(p_y, p_x)) \Rightarrow p_x = p_y \tag{2}
\]
\[
\forall p_x, p_y, p_z : \mathcal{P} \bullet (\mathcal{P}(p_x, p_y) \land \mathcal{P}(p_y, p_z)) \Rightarrow \mathcal{P}(p_z, p_z) \tag{3}
\]

\(^8\)Our notation now is not RSL but a conventional first-order predicate logic notation.
• Proper Underlap, $\mathcal{PU}$,

$\diamond px$ and $py$ are said to properly underlap if
$\diamond px$ and $py$ under-cross and
$\diamond py$ and $px$ under-cross.

\[
\mathcal{PU}(px, py) \triangleq UX(px, py) \land UX(py, px) \tag{4}
\]
4.4.1.2 A Model for the Axioms

• [6] now gives a model for
  ◆ parts: atomic and composite,
  ◆ commensurate with [1] and [4], and
  ◆ their unique identifiers, mereology and attributes
and show that the model satisfies the axioms.

4.4.2. Contributions of [6]

• [6] thus contributes
  ◆ to a domain science,
  ◆ helping to secure a firm foundation for domain engineering.
5. The Experiments [10–24]

• In order to test and tune the domain analysis & description method
  ◆ a great number of experiments were carried out.
  ◆ In our opinion, when applied to manifest domains, they justify
    the calculi reported in [1] and [4].
• Urban Planning [10],
• Documents [11],
• Credit Cards [12],
• Weather Systems [13],
• The Tokyo Stock Exchange [14],
• Pipelines [15],
• Road Transportation [16],
• Web–based Software [17],
• “The Market” [18],
• Container Lines [19] and
• Railways [20, 21, 22, 23, 24].
6. **Summary**

- We have identified a discipline of domain science and engineering.
  - Its first “rendition” was applied to the semantics of programming languages and the development of their compilers [27, CHILL] and [30, Ada].
  - Domain science and engineering, as outlined here, is directed at a wider spectrum of “languages”:
    - the “meaning” of computer application domains
    - and software for these applications.
• Where physicists model facets of the world emphasizing physical, dynamic phenomena in nature, primarily using differential calculi,

• domain scientists cum engineers emphasize logical and both discrete phenomena of man and human institutions primarily using discrete mathematics.

7. Laudatio

• At dinner, tonigh, I shall give a dinner speech.
• It is not about Zhou Chaochen’s scientific life.
• But it is a laudatio
  ◆ expressed in deep love
  ◆ for a wonderful man
  ◆ and our lives together.
• Zhou and Wang Ji: Thanks for my being here.
8. References


Contents

1. Introduction ................................................................. 2
   1.1. Recent Papers .................................................. 4
   1.2. Recent Experiments ........................................... 5
   1.3. My Emphasis on Software Systems ......................... 6
   1.4. How Did We Get to Domain Science & Engineering? .... 9
   1.5. Method & Methodology ....................................... 11
   1.6. Computer & Computing Sciences ......................... 12
2. The Papers ..................................................................... 13
   3.1. A Domain Ontology ............................................... 15
       3.1.1. Parts, Components and Materials ..................... 15
       3.1.2. Unique identifiers ...................................... 26
       3.1.3. Mereology ............................................... 27
       3.1.4. Attributes .............................................. 29
   3.2. From Manifest Parts to Domain Behaviours .................. 32
       3.2.1. The Idea — by means of an example ................. 33
       3.2.2. Atomic Parts ........................................... 34
       3.2.3. Composite Parts ....................................... 35
       3.2.4. Concrete Parts ......................................... 36
       3.2.5. Translation of Part Qualities (...) ......... 37
4. “The Other Papers” ........................................................ 40
   4.1. Domain Facets: Analysis & Description [2, 3] .......... 40
       4.1.1. Overview ............................................... 40
       4.1.2. Contributions of [2, 3] — and Open Problems .... 43
   4.2. From Domain Descriptions to Requirements Prescriptions [7, 8] 44
       4.2.1. Overview ............................................... 44
       4.2.2. Contributions of [7, 8] ............................. 48
   4.3. Formal Models of Processes and Prompts [4, 5] ........... 49
       4.3.1. Overview ............................................... 49
       4.3.2. Contributions of [4] ................................ 51
   4.4. To Every Manifest Domain Mereology a CSP Expression [6] 52
       4.4.1. Overview ............................................... 52
       4.4.2. Contributions of [6] ................................ 56
5. The Experiments [10–24] ............................................... 57
6. Summary ....................................................................... 59
7. Laudatio ....................................................................... 60
8. References ..................................................................... 62