UDC 004.4

D. BJØRNER

DOMAIN SCIENCE AND ENGINEERING FROM COMPUTER SCIENCE TO THE SCIENCES OF INFORMATICS. PART II: SCIENCE

Keywords: *domain, domain description, domain engineering, domain modelling, software development, software engineering.*

1. THE PROBLEM OF THIS PAPER

The problem is twofold. (1) There is a dichotomy between what an informal description, a narrative, refers to, and what a formal description, a formalisation, refers to. (2) And which are our means of description?

Problem (1) is indicated in Fig. 1.

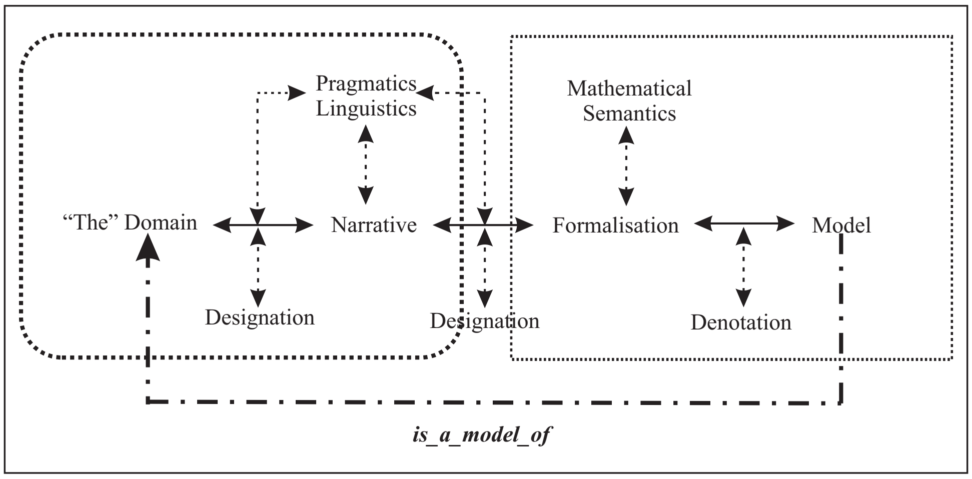

Fig. 1. Formal and informal relations

Issue (1) was amply illustrated in Sec. 2 of Part I of this paper [3]. A narrative, which is expressed in some national, that is, informal language, designates some (i.e., "the") domain. A formalisation (supposedly of some (i.e., "the") domain) denotes a mathematical model. One can claim, as we shall, that a formalisation designates a number of narratives, including "the one" given. One can "narrate" a formalisation. The "beauty" of a formalisation, if any, is its ability to "inspire" designated formalisations that are worth reading, that exhibit a clear, didactic understanding of the domain, and is pedagogical, that is, introduces domain phenomena and notions, gently, one-by-one.

In Sec. 2 of [3] we presented a pair of descriptions – purportedly – of a domain of (a class of) pipeline systems. In the narrative part of the pair such terms as `type`, `value`, `entity`, `function` and `behaviour` were intended to "point to", to refer to phenomena and concepts of the domain, that is, to sets of such phenomena and concepts formed, basically, from these phenomena.

In the formal part of the pair of domain descriptions the corresponding, formal textual phrases, i.e., the syntactic sentences of the formulas, have a semantics, and that semantics, as in RSL, is usually, for formal specifications, some mathematical structures. Therein, lies the first problem: The relation between narrative texts and the domain, “the real world”, can only be an informal one. The relation between the formal texts and their semantics is a formal one. “*And never the twain shall meet!*”[1] One can not establish a formal relation between the informal world of domains and the formal world of mathematical specifications. **The above is an essence of abstract models**.

Then we outline the “describability” problem. **What, of actual domains, can be modelled?** That is, what, of any domain, on one hand, can be narrated, all or some, and on the other hand, can be formalised, all or some? This double question is one of ontology and of epistemology. This section shall discuss issues related to Item (i–ii) of Footnote 3.

1.1. Russell’s Problem

Bertrand Russell, in [49] and in several other writings, brought the following example (abbreviated): “`The present King of France`”. At present there is no designation in any domain of such a person. Thus the sentence does not make sense. In a formalisation we would express this as:

type
    DOMAIN, DESIGNATION
value
    obs_DESIGNATIONs: DOMAIN $\rightarrow$ DESIGNATION-**infset**
    designation: Sentence $\rightarrow$ DOMAIN $\overset{\sim}{\rightarrow}$ DESIGNATION
    existence: Sentence $\rightarrow$ DOMAIN $\rightarrow$ **Bool**
    designation(s)(ω) $\equiv$ **if** existence(s)(ω) **then ... else chaos end**

“`The present King of France`” did have a designation around year 1600. I claim that Russell’s “problem” lies in the dashed (rounded edge) box of the left side of Fig. 1. The simpler-minded computer/computing science problem of syntax and semantics is then that of the rectangular box of the right side of Fig. 1.

1.2. The Problem of This Paper

The problem of this paper should now be a little more clear: It is that of “reconciling” of what is indicated by the two “boxes” of Fig. 1: the classical epistemological universe of discourse of the left side of that figure, and the domain science universe of discourse of the right side of that figure.

Put in different terms: We would somehow like to establish that the horisontal, non-dashed double arrows expresses that the model (to the extreme right) “*is a model*” of the `domain` (to the extreme left) — such as indicated by the bottom left-pointing dashed arrow (with the upside-down[2] *is_a_model_of* label).

In Sec. 2 of part I of this paper [3] we presented a non-trivial example. It served to illustrate the new concepts of *domain engineering*, *domain description* and *domain model*. Section 3 the Part I paper then discussed the concept of domains. In Sec. 2 of the present paper we shall embark on a discussion of our logical and linguistic means for description, of “the kind of ‘*things*’ that can ‘*exist*’ or the things (say in the domain, i.e., “real world”) that they reflect”.

[1] Rudyard Kipling, Barrack-room Ballads, 1892: “Oh, East is East, and West is West, and never the twain shall meet.”
[2] So as to be read from where dashed arrow emanates to where it is incident.

2. A SPECIFICATION ONTOLOGY AND EPISTEMOLOGY[3]

2.1. What Is an Ontology?

2.1.1. Some remarks. By a specification ontology we shall understand a set of mathematical concepts to be used in specifying “something”. By a domain description ontology we shall understand a set of concepts to be used in describing a domain. We shall choose a textual, rather than a diagrammatic (graphical) form for expressing descriptions. The description ontology therefore hinges on a number of textual (i.e., non-diagrammatic) syntactic constructs. These will be covered in Secs. 2.2–2.5 and 2.7–2.8. The pragmatics of description designations (using these syntactic constructs) are, of course, phenomena of the domain. The semantics of description designations are, of course, mathematical constructs. The above remarks, naturally, influence the rest of this more “theoretical/philosophical” part of the paper. That is, we thus hint at, but do not present, a more thorough philosophy of science reasoning, arguments that should support our description ontology.

2.1.2. Description ontology versus ontology description. According to Wikipedia: *Ontology is the philosophical study of* (i) *the nature of being, existence or reality in general*, (ii) *as well as of the basic categories of being and their relations*. Section 2 of [3] emphasized the need for describing domain phenomena and concepts. This section (Sec. 2 of this paper) puts forward a description ontology — (i) *which “natures of being, existence or reality”* and (ii) *which “categories of being and their relations”* — that we shall apply in the description of domain phenomena and concepts.

Yes, we do know that the term ‘description ontology’ can easily be confused with ‘ontology description’ — a term used very much in two computing related communities: AI (artificial intelligence) and WWW (World Wide Web). These communities use the term ‘ontology’ as we use the term ‘domain’ [2, 13, 15, 22–25, 51]. But their use of ‘ontology’ is mostly directed at an analysis of the entities, not so much the operations, events and behaviours. They are also mostly interested in such formal specifications that can be subjected to automatic analysis. Our use of ‘description ontology’ is to analyse and formalise also incomputable [31] phenomena and concepts.

By domain ‘description ontology’ we shall mean a set of notions that are used in describing a domain. So the ontology is one of the description language not of the domain that is being described.

2.2. Categories, Predicates and Observers for Describing Domains

It is not the purpose of this paper to motivate the categories, predicates and observer functions for describing phenomena and concepts. This is done elsewhere [4, 5, 8–10]. Instead we shall more-or-less postulate one approach to the analysis of domains. We do so by postulating a number of meta-categories, meta-predicates and meta-observer functions. They characterise those non-meta categories, predicates and observer functions that the domain engineer cum researcher is suggested to make use of. There may be other approaches [50] than the one put forward in this paper.

[3] Webster Collegiate Dictionary explanation of philosophical terms:
Ontology: (i) a branch of metaphysics concerned with the nature and relations of being;
(ii) a particular theory about the nature of being or the kinds of things that have existence.
(iii) Epistemology: the study or a theory of the nature and grounds of knowledge especially with reference to its limits and validity.

2.2.1. An aside on notation. In this entire section we shall be using two kinds of notation. Both may look like uses of RSL, but they are not. A notation which involves the use of THIS FONT; and a notation which, in some form of mathematics, explain the former. Please note that these meta-functions, those "partially spelled" with THIS FONT are not RSL functions but are mental functions applied by the domain modeller in the analysis of domains.

2.2.2. The hypothetical nature of categories, predicates and observers. In the following we shall postulate some categories of phenomena[4], that is, some meta-types:

categories
0: ALPHA, BETA, GAMMA

What such a clause as the above means is that we postulate that there are the categories ALPHA, BETA, GAMMA of "things" (phenomena and concepts) in the world of domains. That is, there is no proof that such "things" exists. It is just our way of modelling domains. If that way is acceptable to other domain science researchers or domain engineers, fine. In the end, which we shall never reach, those aspects of a, or the domain science, may "survive". If not, well, then they will not "survive" !

2.2.3. Predicates and observers. With the categories just introduced we then go on to postulate some predicate and observer functions. For example:

predicate signatures
1: is_ALPHA: "Thing" $\rightarrow$ **Bool**
2: is_BETA: "Thing" $\rightarrow$ **Bool**
3: is_GAMMA: "Thing" $\rightarrow$ **Bool**
observer signatures
4: obs_ALPHA: "Thing" $\xrightarrow{\sim}$ ALPHA
5: obs_BETA: ALPHA $\xrightarrow{\sim}$ BETA
6: obs_GAMMA: ALPHA $\xrightarrow{\sim}$ GAMMA

So we are "fixing" a logic! Observing, as in [4], an ALPHA "in" (or "from") a "thing" is, of course, not possible, if "that thing" does not "contain" an ALPHA. Hence the observer function is partial: not defined for all "things", yielding **chaos** for such "things" which do not "contain" an ALPHA. Etcetera.

The "Things" clause is a reference to the domain under scrutiny. Some 'things' in that domain are of category[5] ALPHA, or BETA, or GAMMA. Some are not. It is then postulated that from such things of category ALPHA one can observe things of categories BETA or GAMMA. Whether this is indeed the case, i.e., that one can observe these things is a matter of conjecture, not of proof.

2.2.4. Uncertainty. The function signature:
value
    fct: A $\xrightarrow{\mathfrak{R}}$ B
expresses a relation, $\text{fct}_{\mathfrak{R}}$, which one may think of as:
type
    $\text{fct}_{\mathfrak{R}}$: $\text{FCT}_{\mathfrak{R}}$ = (A × B)-**infset**.

Applying fct to an argument `a`, that is, `fct(a)`, may then either "always" result in some specific `b`, in which case fct is a function; or result in **chaos**, that is, `fct` is not defined for that argument `a`, that is: `(a,b)` $\notin \text{fct}_{\mathfrak{R}}$ for any `b`; or sometimes result in some `b`, sometimes in another `b′`, etc., that is, $\text{fct}_{\mathfrak{R}} = \{$`(a,b), (a,b′), (a,b″),` ... $\}$.

[4] These observable phenomena or abstract concepts are, in general, entities, more specifically either simple entities, actions, events or behaviours.
[5] We use the term 'category' in lieu of either of the term 'type', 'class', 'set'. By here using the term 'category' we do not mean category in the mathematical sense of category theory.

2.2.5. Meta-conditions. Finally we may sometimes postulate the existence of a meta-axiom:

meta condition:

Predicates over ALPHA, BETA and GAMMA

Again, the promulgation of such logical meta-expressions are just conjectures, not the expression of "eternal" truths which would then be laws of the (description of the) domain.

2.2.6. Discussion. So, all in all, we suggest four kinds of meta-notions:

- **`categories`**,
- **`is_Category`** and **`obs_Property`** predicates,
- **`obs_Category`** and **`obs_Attribute`** observers, and
- **`meta-conditions`**, i.e., axioms.

The **category** [**type**] A, B, ..., the predicate is_A, is_B, ... functions, the observer obs_A, obs_B, ... functions, and the **meta-condition** [**axiom**] predicate notions derive from McCarthy's *analytic syntax* [35].

Discussion. Thus the formal specification and the high level programming languages' use, that is, the software designers' use of **type** clauses, predicate functions and observer (in the form of selector) functions shall be seen in the context of specifications, respectively program code dealing with computable quantities and decomposing and constructing such quantities.

The proposal here, of suggesting that the domain engineer cum researcher makes us of c**ategories**, **predicates**, **observers** and **meta-conditions** is different. In domain descriptions an existing "universe of discourse" is being analysed. Perhaps the **categories**, **predicates**, **observers** and **meta-conditions** makes sense, perhaps the domain engineer cum researcher can use these descriptional "devices" to "compose" a consistent and relative complete "picture", i.e., description, of the domain under investigation.

Either the software designers' use of formal specification or programming language constructs is right or it is wrong, but the domain engineer cum researchers' use is just an attempt, a conjecture. If the resulting domain description is inconsistent, then it is wrong. But it can never be proven right. Right in the sense that it is **the** right description. As in physics, it is just a conjecture. There may be refutations of domain models.

2.3. Entities

What we shall describe is what we shall refer to as entities. In other words, there is a category ENTITY, and there is a meta-logical predicate is_ENTITY. The is_ENTITY predicate applies to "whatever" in the domain, whether an entity or not, and "decides", i.e., is postulated to analyse whether that "thing" is an entity or not:

predicate signature:

is_ENTITY: "Thing" $\rightarrow$ **Bool**

meta condition:

$\forall$ e:ENTITY • is_ENTITY(e)

Discussion. When we say "things", or entities, others may say 'individuals', 'objects', or use other terms.

The meta-predicate is_ENTITY provides a rather "sweeping" notion, namely that someone, the domain engineer, an oracle or other, can decide whether "something" is to be described as a phenomenon or concept of the domain.

By introducing the predicate is_ENTITY we have put the finger on what this section is all about, namely “*what exists?*” and “*what can be described?*” We are postulating a description ontology. It may not be an adequate one. It may have flaws. But, for the purposes of raising some issues of epistemological and ontological nature, it is adequate.

2.3.1. Entity categories. We postulate four entity categories:

category:
SIMPLE_ENTITY, ACTION, EVENT, BEHAVIOUR

Some **phenomena** or **concepts** are simple entities.

Discussion. That is, the domain describer is free to decide that a phenomenon or a concept is a simple entity. The decision, of course, has implications. It may be a good modelling decision; or it may be a not-so-good modelling decision. There is nothing inherent, or innate, we claim, in the world, that is, in some chosen domain of that world, which mandates that something is a simple entity.

Simple entity phenomena are the things we can point to, touch and see. They are manifest. Other phenomena, for example those we can hear, smell, taste, or measure by physics (including chemistry) apparatus are properties (attributes) of simple entity phenomena. **Concepts** are abstractions about phenomena and/or other concepts.

Discussion. We have made a rather important decision here; and that is: we make a distinction between simple entities and their properties (attributes). We claim that simple entity phenomena are manifest and that their attributes are a way of (not necessarily fully) characterising such simple entities. We shall further claim that one cannot remove an attribute (a property) from a simple entity and, at the same time, retaining that simple entity (less the “removed attribute”). So attributes are indispensable articulations (ingredients, measures) of simple entities. A “free-standing” property is not a simple entity.

Examples. The height, age, gender, weight, hair color, profession, etc., of a person are attributes. The height value 179 cm is not a separately manifest simple entity.

Attributes can be classified as either static or dynamic. Static attributes have values that are constant, cannot change. Dynamic attributes have values that are variable, can change.

Discussion. Again it is a decision to be made by the describer: whether an attribute is static or dynamic. It depends, one can claim, on the temporal “perspective” that the modeller decides to apply. Seen with a narrow, detailed, say micro-second interval time granularity an attribute may be considered dynamic, whereas seen from a 24 hour granularity an attribute may be considered static.

Examples. A persons passport data, height, for example, is considered static; valid, approximately, for the validity period of the passport. A person’s weight is, by a medical, say diabetes doctor, considered dynamic.

A subset of simple domain entities form a **state. Actions** are the result of applying functions to simple domain entities and changing the **state**. What is changed are the attribute values of simple (state) entities. Actions are observable through the observation of the occurrence of the ‘before’ and ‘after’ states. The functions or relations that relate before and after states are not observable. They are our way of “explaining” the actions. If you wish to consider them as simple entities then they are atomic have no name, but do have a type, the function signature. Actions are caused by domain behaviours.

Events are **state** changes that satisfy a predicate on the 'before' and 'after states'. Events are observable through their "taking place", that is, by observing, as if they were actions, their 'before' and 'after states', but also by, somehow, observing that they are not caused by domain behaviours, well, possibly then by behaviours "outside" the domain being considered. The above represents a "narrow" concept of events. A less narrow concept would characterise some domain actions as events; we might call them "interesting" action events.

Behaviours are sets of sequences (of sets of) actions and events. Behaviours are observable – through the observation of the constituent actions and events.

Below we shall have "much more" to say about these four categories of entities.

category:

ENTITY = SIMPLE_ENTITY ∪ ACTION ∪ EVENT ∪ BEHAVIOUR

Discussion. The four categories of entities may overlap.

With each of the four categories there is a predicate:

predicate signature:

is_SIMPLE_ENTITY "Thing" $\overset{\Re}{\to}$ **Bool**

is_ACTION "Thing" $\overset{\Re}{\to}$ **Bool**

is_EVENT "Thing" $\overset{\Re}{\to}$ **Bool**

is_BEHAVIOUR "Thing" $\overset{\Re}{\to}$ **Bool**

Each of the above four predicates require that their argument `t`:"Thing" satisfies:

is_ENTITY(`t`)

The use of $\overset{\Re}{\to}$ shall illustrate the uncertainty that may befell the domain modeller. We shall henceforth "boldly" postulate functionality, i.e., →, of the is_SIMPLE_ENTITY, is_ACTION, is_EVENT and is_BEHAVIOUR functions.

The ∪ "union" is inclusive:

meta condition:

∀ t: "Thing" • is_ENTITY(t) ⇒
  is_SIMPLE_ENTITY(t) ∨ is_ACTION(t) ∨ is_EVENT(t) ∨ is_BEHAVIOUR(t)

2.4. Simple Entities

We postulate that there are **atomic** simple entities; that there are therefrom distinct composite simple entities; that a simple entity is indeed either atomic or composite; that atomic simple entities cannot meaningfully be described as consisting of proper other simple entities; but that composite simple entities indeed do consist of proper other simple entities. It is us, the observers, who decide to abstract a simple entity as either being atomic or as being composite.

That is:

category:

SIMPLE_ENTITY = ATOMIC ∪ COMPOSITE

observer signature:

is_ATOMIC: SIMPLE_ENTITY → **Bool**

is_COMPOSITE: SIMPLE_ENTITY → **Bool**

meta condition:

ATOMIC ∩ COMPOSITE = {}

∀ s: "Things":SIMPLE_ENTITY •
  is_ATOMIC(s) ≡~is_COMPOSITE(s)

Discussion. We put in brackets, in the text paragraph before the above formulas, there-from distinct. One may very well discuss this constraint — **`are there simple entities that are both atomic and composite?`** — and that is done by Bertrand Russell in his 'Philosophy of Logical Atomism' [49].

2.4.1. Discrete and continuous entities. We postulate two forms of SIMPLE_ENTITIES: DISCRETE, such as a railroad net, a bank, a pipeline pump, and a securities instrument, and CONTINUOUS, such as oil and gas, coal and iron ore, and beer and wine.

category:

SIMPLE_ENTITY =

= DISCRETE_SIMPLE_ENTITY $\cup$ CONTINUOUS_SIMPLE_ENTITY

predicate signatures:

is_DISCRETE_SIMPLE_ENTITY: SIMPLE_ENTITY $\rightarrow$ **Bool**

is_CONTINUOUS_SIMPLE_ENTITY: SIMPLE_ENTITY $\rightarrow$ **Bool**

meta condition:

(is it desirable to impose the following?)

$\forall$ s: SIMPLE_ENTITY •

is_DISCRETE_SIMPLE_ENTITY(s) $\equiv\sim$CONTINUOUS_SIMPLE_ENTITY(s)

Discussion. In the last lines above we raise the question whether it is ontologically possible or desirable to be able to have simple entities which are both discrete and continuous. Maybe we should, instead, express an axiom which dictates that every simple entity is at least of one of these two forms.

2.4.2. Attributes. Simple entities are characterised by their attributes: attributes have name, are of type and has some value; no two (otherwise distinct) attributes of a simple entity has the same name.

category:

ATTRIBUTE, ATTRIBUTE_NAME, TYPE, VALUE

observer signature:

obs_ATTRIBUTEs: SIMPLE_ENTITY $\rightarrow$ ATTRIBUTE-**set**

obs_ATTRIBUTE_NAMEs: SIMPLE_ENTITY $\rightarrow$ ATTRIBUTE_NAME-**set**

obs_ATTRIBUTE_NAME: ATTRIBUTE $\rightarrow$ ATTRIBUTE_NAME

obs_TYPE: ATTRIBUTE $\rightarrow$ TYPE

obs_VALUE: ATTRIBUTE $\rightarrow$ VALUE

meta condition:

$\forall$ s:SIMPLE_ENTITY •

$\forall$a, a′: ATTRIBUTE • {a, a′}$\subseteq$obs_ATTRIBUTEs(s)

$\wedge$ a $\neq$ a′ $\Rightarrow$ obs_ATTRIBUTE_NAME(a) $\neq$ obs_ATTRIBUTE_NAME(a′)

Examples of attributes of atomic simple entities are: (i) A pipeline pump usually has the following attributes: **`maximum pumping capacity, current pumping capacity, whether for oil or gas, diameter (of pipes to which the valve connects)`**, etc. (ii) Attributes of a person usually includes **`name, gender, birth date, central registration number, address, marital state, nationality`**, etc.

Examples of attributes of composite simple entities are: (iii) A railway system usually has the following attributes: **`name of system, name of geographic areas of location of rail nets and stations, whether a public or a private company, whether fully, partly or not electrified`**, etc. (iv) Attributes of a bank usually includes: **`name of bank, name of geographic areas of location of bank branch offices, whether a commercial portfolio bank or a high street, i.e., demand/deposit bank`**, etc.

We do not further define what we mean by attribute names, types and values. Instead we refer to [37] and [20] for philosophical discourses on attributes.

2.4.3. Atomic simple entities: Attributes. Atomic simple entities are characterised only by their attributes.

Discussion. We shall later cover a notion of domain actions, that is functions being applied to entities, including simple entities. We do not, as some do for programming languages, "lump" entities and functions, etc. into what is there called 'objects'.

2.4.4. Composite simple entities: Attributes, sub-entities and mereology. Composite simple entities are characterised by three properties: (i) their **attributes**, (ii) a proper set of one or more **sub-entities** (which are simple entities) and (iii) a **mereology** of these latter, that is, how they relate to one another, i.e., how they are composed.

Sub-entities

Proper sub-entities, that is simple entities properly contained, as immediate **parts** of a composite simple entity, can be observed (i.e., can be postulated to be observable):

observer signature:

  obs_SIMPLE_ENTITIES: COMPOSITE $\to$ SIMPLE_ENTITY-**set**

Mereology

Mereology is the theory of **part-hood** relations: of the relations of part to whole and the relations of **part** to **part** within a **whole**. Suffice it to suggest some mereological structures:

• **Set Mereology:** The individual sub-entities of a composite entity are "un-ordered" like elements of a set. The obs_SIMPLE_ENTITIES function yields the set elements.

predicate signature:

  is_SET: COMPOSITE $\to$ **Bool**

• **Cartesian Mereology:** The individual sub-entities of a composite entity are "ordered" like elements of a Cartesian (grouping). The function obs_ARITY yields the arity, 2 or more, of the simple Cartesian entity. The function obs_CARTESIAN yields the Cartesian composite simple entity.

predicate signature:

  is_CARTESIAN: COMPOSITE $\to$ **Bool**

observer signatures:

 obs_ARITY: COMPOSITE $\overset{\sim}{\to}$ **Nat**

   pre: obs_ARITY(s): is_CARTESIAN(s)

 obs_CARTESIAN: COMPOSITE $\overset{\sim}{\to}$

      SIMPLE_ENTITY $\times$ ... $\times$ SIMPLE_ENTITY

   pre obs_CARTESIAN(s): is_CARTESIAN(s)

meta condition:

 $\forall$ c:SIMPLE_ENTITY •

   is_COMPOSITE(c)$\wedge$ is_CARTESIAN(c) $\Rightarrow$

    obs_SIMPLE_ENTITIES(c) = **elements of** obs_CARTESIAN(c)

    $\wedge$ **cardinality of** obs_SIMPLE_ENTITIES(c) = obs_ARITY(c)

We just postulate the **elements of** and the **cardinality of** meta-functions. Although one may have that distinctly positioned elements of a Cartesian to be of the same type and even the same value, they will be distinct – with distinctness "deriving" from their distinct positions.

• **List Mereology:** The individual sub-entities of a composite entity are “ordered” like elements of a list (i.e., a sequence). Where Cartesians are fixed arity sequences, lists are variable length sequences.

```
predicate signature:
      is_LIST: COMPOSITE → Bool
observer signatures:
      obs_LIST: COMPOSITE $\xrightarrow{\sim}$ list of SIMPLE_ENTITY
         pre obs_LIST(s): is_LIST(s)
      obs_LENGTH: COMPOSITE $\xrightarrow{\sim}$ Nat
   pre obs_LENGTH(s): is_LIST(s)

meta condition: ∧
 ∀ s:SIMPLE_ENTITY•
       is_COMPOSITE(s) ∧ is_LIST(s) ⇒
          obs_SIMPLE_ENTITIES(s) = elements of obs_LIST(s) ∧
          cardinality of elements of obs_LIST(s) = LENGTH(s) ∧
          ∀ e,e′ • {e,e′} ⊆ elements of ⇒
               obs_LIST(s) ⇒ obs_TYPE(e)=obs_TYPE(e′)
```

We also just postulate the **list of**, **elements of** and the **cardinality of** meta-functions. Again, as for Cartesians, we shall postulate that although distinct elements of a list (which are all of the same type) may have the same value – they are distinct due to their distinct position in the list, that is, their adjacency to a possible immediately previous and to a possibly immediately following element.

• **Graph Mereology:** The individual sub-entities of a composite entity are “ordered” like elements of a graph, i.e., a net, of elements. Trees, lattices, cycles and other structures are just special cases of graphs. Any (immediate) sub-entity of a composite simple entity of GRAPH mereology may be related to any number of (not necessarily other) (immediate) sub-entities of that same composite simple entity GRAPH in a number of ways: it may immediately PRECEDE, or immediate SUCCEED or be BIDIRECTIONALLY_LINKED with these (immediate) sub-entities of that same composite simple entity. In the latter case some sub-entities PRECEDE a SIMPLE_ENTITY of the GRAPH, some sub-entities SUCCEED a SIMPLE_ENTITY of the GRAPH, some both.

```
predicate signature:
   is_GRAPH: COMPOSITE → Bool
observer signatures:
   obs_GRAPH: COMPOSITE $\xrightarrow{\sim}$ GRAPH
            pre obs_GRAPH(g): is_GRAPH(g)
   obs_PRECEEDING_SIMPLE_ENTITIES:
         COMPOSITE × SIMPLE_ENTITY → SIMPLE_ENTITY-set
            pre obs_PRECEEDING_SIMPLE_ENTITIES(c,s):
                     is_GRAPH(c) ∧ s ∈ obs_SIMPLE_ENTITIES(c)
   obs_SUCCEEDING_SIMPLE_ENTITIES:
         COMPOSITE × SIMPLE_ENTITY → SIMPLE_ENTITY-set
            pre obs_PRECEEDING_SIMPLE_ENTITIES(c,s):
                     is_GRAPH(c) ∧ s ∈ obs_SIMPLE_ENTITIES(c)
```

meta condition:

```
  $\forall$ c: SIMPLE_ENTITY • is_COMPOSITE(c) $\wedge$ is_GRAPH(c)
        $\Rightarrow$ **let** ss = SIMPLE_ENTITIES(c) **in**
          $\forall$ s′: SIMPLE_ENTITY • s′ $\in$ ss
            $\Rightarrow$ obs_PRECEEDING_SIMPLE_ENTITIES(c)(s′) $\subseteq$ ss
              $\wedge$ obs_SUCCEEDING_SIMPLE_ENTITIES(c)(s′) $\subseteq$ ss
          **end**
```

2.4.5. Discussion. Given a “thing”, *s*, which satisfies is_SIMPLE_ENTITY(s), the domain engineer can now systematically analyse this “thing” using any of the is_ATOMIC(s), is_COMPOSITE(s), is_SET(s), is_CARTESIAN(s), is_LIST(s), is_GRAPH(s), etc., predicates and using also the observer functions sketched above.

Given any SIMPLE_ENTITY the domain engineer can now analyse it to find out whether it is an ATOMIC or a COMPOSITE entity. An, in either case, the domain engineer can analyse it to find out about its ATTRIBUTES. If the SIMPLE_ENTITY is_COMPOSITE then its SIMPLE_ENTITIES and their MEREOLOGY can be additionally ascertained. In summary: If ATOMIC then ATTRIBUTES can be analysed. If COMPOSITE then ATTRIBUTES, SIMPLE_ENTITIES and MEREOLOGY can be analysed.

Please note that these meta-functions, those “partially spelled” with THIS FONT, are not RSL functions but are mental functions applied, conceptually, i.e., “by the brain” of the domain modeller in the analysis of domains.

2.4.6. Practice. How do we interpret this section, Sec. 2.4, on simple entities? We practice it by analysing the domain according to the principles laid down in this section, Sec. 2.4, by discovering simple entities of the domain, by discovering and writing down, for example in RSL, their sorts (i.e., abstract types) or their concrete types, by possibly also discovering (postulating, conjecturing) and writing down constraints, that is axioms or well-formedness predicates over these sorts and types. That is, we are not using these “funny” names, such as SIMPLE_ENTITY, ATOMIC, COMPOSITE, DISCRETE_SIMPLE_ENTITY, CONTINUOUS_SIMPLE_ATTRIBUTE, NAME, TYPE, VALUE, CARTESIAN, ARITY, LIST, LENGTH, GRAPH, PRECEDING_SIMPLE_ENTITIES, SUCCEEDING_SIMPLE_ENTITIES, etc., nor their `is_` or `obs_` forms. The example of Sec. 2 of part I [3] of this paper has given several examples of sort and type definitions as well as of constraint definitions.

2.5. Actions

2.5.1. Definition of actions. By a STATE we mean a set of one or more SIMPLE_ENTITIES. By an ACTION we shall understand the application of a ***FUNCTION*** to (a set of, including the state of) SIMPLE_ENTITIES such that a STATE change occurs.

2.5.2. Non-observables. The mathematical concept of a function, explained as “*that thing which when applied to something(s) called its arguments yields (i.e., results in) something called its results*”, that concept, is an elusive one. No-one has ever “seen”, “touched”, “heard” or otherwise sensed a function. Functions, as a concept, is purely a mathematical construction possessing a number of properties and introduced here in order to explain what is going on in any domain. We shall not be able to observe functions. To highlight this point we use this ***SPELLING OF FUNCTIONS***.

2.5.3. Atomic and composite actions. One can, in principle, talk about composite actions, that is, actions composed from two or more (other) actions, and so forth, until one, the domain analyser, determines that the “smallest” actions are no longer composite, but atomic. We shall, in this paper, consider composite actions as being, instead, behaviours. Whether a composite action, had we decided to maintain the composite/atomic action distinction, was then structured, as for simple entities, as a set, a Cartesian or list, or an acyclic "graph" (a lattice) of actions, themselves composite or atomic would then have lead to the same kind of structure analysis as we apply to behaviours, see Sec. 2.8.2.

2.5.4. Observers &c. We postulate that the domain engineer can indeed decide, that is, conjecture, whether a "thing", which is an ENTITY is an atomic ACTION.

category:
ACTION, ***FUNCTION***, STATE
predicate signature:
is_ACTION: ENTITY $\rightarrow$ **Bool**

Given an ENTITY of category ACTION one can observe, i.e., conjecture the ***FUNCTION*** (being applied), the ARGUMENT CARTESIAN of SIMPLE_ENTITIES to which the ***FUNCTION*** is being applied, and the resulting change STATE change. None of the elements of the CARTESIAN ARGUMENT are allowed to be SIMPLE STATE ENTITIES.

category:
STATE = SIMPLE_ENTITY
FUNCTION = ARGUMENT $\times$ STATE $\rightarrow$ STATE
ARGUMENT = {|s:SIMPLE_ENTITY•is_CARTESIAN(s)|}
observer signatures:
obs_ACTION: ENTITY $\rightarrow$ ACTION
obs_ARGUMENT: ACTION $\rightarrow$ ARGUMENT
obs_INPUT_STATE: ACTION $\rightarrow$ STATE
obs_RESULT_STATE: ACTION $\rightarrow$ STATE

2.5.5. Practice. How do we interpret this section, Sec. 2.5, on actions? We practice it by analysing the domain according to the principles laid down in this section, Sec. 2.5, by discovering actions of the domain, by discovering and writing down, for example in RSL, their signatures, by possibly also discovering (postulating, conjecturing) and writing down their definitions. That is, we are not using these "funny" names, such as ACTION, STATE, ARGUMENT, INPUT_STATE, RESULT_STATE nor there `is_` or `obs_` forms. The pipeline example of Sec. 2 of part I of this paper [3] has given several examples of action signatures and definitions.

2.6. "Half-way" Discussion[6]

Some pretty definite assertions were made above: **`We postulate that the domain engineer can indeed decide whether a "thing", which is an`** ENTITY **`is an`** ACTION. And that one can conjecture the ***FUNCTION***, the ARGUMENT and the RESULT of an ACTION. We do not really have to phrase it that deterministically. It is enough to say: One can speak of actions, functions, their arguments and their results. Ontologically we can do so. Whether, for any specific simple entity we can decide whether it is an actions is, in a sense, immaterial: we can always postulate that it is an action and then our analysis can be based on that hypothesis. This discussion applies inter alia to all of the entities being introduced here, together with their properties.

The domain engineer cum researcher can make such decisions as to whether an entity is a simple one, or an action, or an event or a behaviour. And from such a decision that domain engineer cum researcher can go on to make decisions as to

[6] "Half-way": after simple entities and actions and before events and behaviours.

whether a simple entity is discrete or continuous, and atomic or composite, and then onto a mereology for the composite simple entities. Similarly the domain engineer cum researcher can make decisions as to the function, arguments and results of an action. All these decisions do not necessarily represent the “truth”. They hopefully are not “falsities”. At best they are abstractions and, as such, they are approximations.

2.7. Events

Like we did for simple entities we distinguish between atomic and composite events.

2.7.1. Definition of atomic events. By an EVENT we shall understand a pair, (σ,σ'), of STATEs, a time, or time interval, *t*: TIME (at or during which event occurs), and an EVENT PREDICATE, p : ***P***, such that p(σ,σ')(s), yields **true**.

The difference between an ACTION and an EVENT is two things: the EVENT ACTION need not originate within the analysed DOMAIN, and the EVENT PREDICATE is trivially satisfied by most ACTIONs which originate within the analysed DOMAIN.

2.7.2. Examples of atomic events. Examples of atomic events, that is, of predicates are: a bank goes “bust” (e.g., looses all its monies, i.e., bankruptcy), a bank account becomes negative, (unexpected) stop of gas flow and iron ore mine depleted. Respective stimuli of these events could be: (massive) loan defaults, a bank client account is overdrawn, pipeline breakage, respectively over-mining. The example of Sec. 2 of part I of this paper [3] has not given very many examples. Some are: an oil well runs dry (atomic event); a valve gets stuck in closed position, causing further events; a gas pipe breaks causing a fire causing gas to flow, etc. (composite event); etcetera.

2.7.3. Composite events. Definition. A Composite event is composed from a sequence of two or more events: The ‘after’ state of one or more events become the ‘before’ event of an immediately subsequent event.

Example of Composite Event. We consider the first birth of a child to a mother and a father as a composite event: the child is born, is a “first” event of the composition; the **`mother-`** and **`fatherhood`** of the “newborn” parents is a “second” event, following from the fist, but taking place “at the same time as the child **`is born`** event; with the grandparents perhaps, for the first time becoming grandparent, being a “third” event, etcetera. We shall, in this paper, consider composite events as being, instead, behaviours. Whether a composite event, had we decided to maintain the composite/atomic event distinction, was then structured, as for simple entities, as a set, a Cartesian or list, or as an acyclic “graph” (a lattice) of actions, themselves composite or atomic would then have lead to the same kind of structure analysis as we apply to behaviours, see Sec. 2.8.2.

2.7.4. Observers &c. We postulate that the domain engineer from an atomic EVENT can observe the TIME, the BEFORE_STATE, the AFTER_STATE and the EVENT_PREDICATE. As said before: the domain engineer cum researcher can decide on these abstractions, these approximations.

category:

TIME

$\boldsymbol{P}$ = (STATE × STATE) × TIME → **Bool**

observer signatures:

obs_TIME: EVENT → TIME

obs_BEFORE_STATE: EVENT → STATE

obs_AFTER_STATE: EVENT → STATE

obs_EVENT_PREDICATE: EVENT → $\boldsymbol{P}$

meta condition:

```
    $\forall$ e:EVENT •
        $\exists$ t: TIME, $\sigma,\sigma'$: STATE •
            BEFORE_STATE(e) = $\sigma$ $\wedge$
            AFTER_STATE(e) = $\sigma'$ $\wedge$
            $\exists$ p: ***P*** • p($\sigma,\sigma'$)(t)
```

2.7.5. Practice. How do we interpret this section, Sec. 2.7, on events? We practice it by analysing the domain according to the principles laid down in this section, Sec. 2.7, by discovering events of the domain, that is, by discovering and writing down, for example in RSL, the "defining" pre/post condition predicates, by discovering and writing down, for example in RSL, their signatures (as if they were domain actions), by possibly also discovering (postulating, conjecturing) and writing down their definitions (as if they were domain actions). Events, as we shall see in the next section, Sec. 2.8 are, in our treatment, closely linked to behaviours. Therefore we describe events as special actions of behaviours. That is, we are not using these "funny" names, such as EVENT, STIMULUS, EVENT_PREDICATE, BEFORE_STATE, AFTER_STATE nor their **`is_`** or **`obs_`** forms.

2.8. Behaviours

2.8.1. A Loose characterisation. By a BEHAVIOUR we shall understand a set of sequences of ACTIONs and EVENTs. We make a distinction between the domain being analysed and described and the "surroundings", the environment of that domain. With the environment we associate, or rather postulate a behaviour. This atomic or composite environment behaviour is not described further than postulating a number of interactions between the environment behaviour and the domain behaviour(s). The domain behaviour(s) cover different, "overlapping" or "disjoint", parts of the domain. Their interactions with an, or the, environment behaviour, i.e., the events between domain and environment behaviours:

• when modelled in a CSP-like textual notation [30], are expressed as **`CSP channel (ch) inputs (ch?)/outputs (ch!v):`** inputs from the environment, outputs to the environment;

• when modelled in a Petri Net-like graphic notation [47], are expressed by some further designated **`environment places`** that share transitions with **`domain places`**[7]; and

• when modelled in a MSC-like graphic notation [32], are expressed as message inputs/outputs: the, usually horisontal, lines that "connect" the, usually vertical, instance, or behaviour lines.

Similar directions as to modelling can be given for Statechart-like descriptions [28]. It may now be so that some EVENTs in two or more such sequences have their STATEs and PREDICATEs express, for example, mutually exclusive synchronisation and communication EVENTs between these sequences which are each to be considered as simple SEQUENTIAL_BEHAVIOURs. Other forms than mutually exclusive synchronisation and communication EVENTs, that "somehow link" two or more behaviours, can be identified.

2.8.2. Atomic and composite behaviours. It really does not make sense to talk about atomic behaviours unless one insists, and then an atomic behaviour should be taken as an (atomic) ACTION or as an (atomic) EVENT. We will not do so. The "nearest" to an atomic behaviour then becomes a SIMPLE_SEQUENTIAL_BEHAVIOUR, see next.

[7] The sets of **`environment places`** and **`domain places`** are disjoint and covers all Petri Net places.

BEHAVIOUR_INTERACTION_LABELs

We abstract from the orderly example of synchronisation and communication given above and introduce a further un-explained notion of (synchronisation and communication) BEHAVIOUR_INTERACTION_LABELs and allow BEHAVIOURs to now just be sets of sequences of ACTIONs and BEHAVIOUR_INTERACTION_LABELs, such that any one simple sequence has unique labels. BEHAVIOUR_INTERACTION LABELs are just a representation of EVENTSs.

We can classify some BEHAVIOURs.

SIMPLE_SEQUENTIAL_BEHAVIOURs

SIMPLE_SEQUENTIAL_BEHAVIOURs are sequences of ACTIONs and BEHAVIOUR_INTERACTION_LABELs.

```
value
   is_SIMPLE_SEQUENTIAL_BEHAVIOUR: BEHAVIOUR → Bool
   obs_ACTIONandBEHAVIOUR_INTERACTION_LABEL:
   SIMPLE_SEQUENTIAL_BEHAVIOUR →
        (ACTION | BEHAVIOUR_INTERACTION_LABEL)*
     pre obs_ACTIONandBEHAVIOUR_INTERACTION_LABEL(b):
          is_SIMPLE_SEQUENTIAL_BEHAVIOUR(b)
```

SIMPLE_CONCURRENT_BEHAVIOURs

SIMPLE_CONCURRENT_BEHAVIOURs are
sets of SIMPLE_SEQUENTIAL_-BEHAVIOURs.

```
value
   is_SIMPLE_CONCURRENT_BEHAVIOUR: BEHAVIOUR → Bool
   obs_SIMPLE_CONCURRENT_BEHAVIOURs:
      SIMPLE_CONCURRENT_BEHAVIOUR →
         SIMPLE_SEQUENTIAL_BEHAVIOUR-set
      pre SIMPLE_CONCURRENT_BEHAVIOURs(b):
         is_SIMPLE_CONCURRENT_BEHAVIOUR(b)
```

COMMUNICATING_CONCURRENT_BEHAVIOURs

COMMUNICATING_CONCURRENT_BEHAVIOURs are sets of
SIMPLE_CONCURRENT_BEHAVIOURs where every
SIMPLE_CONCURRENT_BEHAVIOUR share at least one
BEHAVIOUR_INTERACTION_LABEL with some other
SIMPLE_CONCURRENT_BEHAVIOUR in the
COMMUNICATING_CONCURRENT_BEHAVIOUR.

```
value
   is_COMMUNICATING_CONCURRENT_BEHAVIOUR: BEHAVIOUR → Bool
   obs_SIMPLE_CONCURRENT_BEHAVIOURs:
      COMMUNICATING_CONCURRENT_BEHAVIOUR →
         SIMPLE_CONCURRENT_BEHAVIOUR-set
      pre obs_SIMPLE_CONCURRENT_BEHAVIOURs(b):
         is_COMMUNICATING_CONCURRENT_BEHAVIOUR(b)
```

We say that two or more such COMMUNICATING_CONCURRENT_BEHAVIOURs *synchronise & communicate* when all distinct BEHAVIOURs “sharing” a (same) label have all reached that label.

Many other composite behaviours can be observed. For our purposes it suffice with having just identified the above. So far we have restricted ourselves to linear, textual descriptions of entities. For behaviours there are some diagrammatic, or graphical description forms that more vividly capture such properties as instantaneous concurrent actions in otherwise distinct behaviours. We are thinking of such description tools as MSC (Message Sequence Charts) [32], LSC (Live Sequence Charts) [29], Petri Nets [47], Statecharts [28], etc.

SIMPLE_ENTITIES, ACTIONs and EVENTs can be described without reference to time. BEHAVIOURs, in a sense, take place over time.[8] It will bring us into a rather long discourse if we are to present some predicates, observer functions and axioms concerning behaviours — along the lines such predicates, observer functions and axioms were present, above, for SIMPLE_ENTITIES, ACTIONs and EVENTs. We refer instead to Johan van Benthem's seminal work on the **`The Logic of Time`** [52]. In addition, more generally, we refer to A.N. Prior's [42–46] and McTaggart's works [36, 17, 48]. The paper by Wayne D. Blizard [11] proposes an axiom system for time-space.

2.8.3. Practice. How do we interpret this section, Sec. 2.8, on behaviours? We practice it by analysing the domain according to the principles laid down in this section, Sec. 2.8, by discovering behaviours of the domain, that is, by discovering and writing down, for example in RSL, the signatures of these behaviours (e.g., as CSP processes), by possibly also discovering (postulating, conjecturing) and writing down their definitions (as sequences of domain actions and events — the latter modelled as CSP communications between behaviours). Events are now, for CSP-based descriptions, the input/output synchronisations and communications of CSP. For MSCs they are the "horisontal lines" that stretch between the vertical behaviour lines. For Petri Nets they are the usually "longish" rectangle transitions that are triggered by tokens present on all incoming "lines" (and transfer tokens from input place holders to output place holders). That is, we are not using all these "funny" names such as: BEHAVIOUR, SEQUENTIAL_BEHAVIOUR, SIMPLE_SEQUENTIAL_BEHAVIOUR, CONCURRENT–BEHAVIOUR, COMMUNICATING_CONCURRENT_ BEHAVIOUR, BEHAVIOUR_INTERACTION_LABELs, etcetera.

2.9. Impossibility of Definite Mereological Analysis

It would be nice if there was a more-or-less obvious way of "deciphering" the mereology of an entity. In the **`Set`**, **`Cartesian`**, **`List`**, **`Map`**, and **`Graph`** subsections above we may have left the impression with the reader that is a more-or-less systematic way of uncovering the mereology of a composite entity. That is not the case: there is no such obvious way, there is no **`Magic Wand`**! It is a matter of both discovery, ingenuity and choice between seemingly alternative mereologies, and it is also a matter of choice of abstraction.

2.10. What Exists and What Can Be Described?

In the previous section we have suggested a number of *categories*[9] of entities,

[8] If it is important that ACTIONs take place over time, that is, are not instantaneous, then we can just consider ACTIONs as very simple SEQUENTIAL_BEHAVIOURs not involving EVENTs.

[9] Some categories: ENTITY, SIMPLE_ENTITY, ACTION, EVENT, BEHAVIOUR, ATOMIC, COMPOSITE, DISCRETE, CONTINUOUS, ATTRIBUTE, ATTRIBUTE_NAME, TYPE, VALUE, SET, CARTESIAN, LIST, MAP, GRAPH, ***FUNCTION***, STATE, ARGUMENT, STIMULUS, EVENT_PREDICATE, BEFORE_STATE, AFTER_STATE, SEQUENTIAL_BEHAVIOUR, BEHAVIOUR_INTERACTION_LABEL, SIMPLE_SEQUENTIAL_BEHAVIOUR, SIMPLE_CONCURRENT_BEHAVIOUR, COMMUNICATING_CONCURRENT_BEHAVIOUR, etc.

a number of *predicate*[10] and *observer*[11] functions and a number of *meta conditions* (i.e., axioms). These concepts and their relations to one-another, suggest an ontology for describing domains. It is now very important that we understand these **`categories`**, **`predicates`**, **`observers`** and **`axioms`** properly.

3. DESCRIPTION VERSUS SPECIFICATION LANGUAGES

Footnotes 9–11 summarised a number of main concepts of an ontology for describing domains. The categories and predicate and observer function signatures are not part of a formal language for descriptions. The identifiers used for these categories are intended to denote the real thing, classes of entities of a domain. In a philosophical discourse about describability of domains one refers to the real things. That alone prevents us from devising a formal specification language for giving syntax and semantics to a specification, in that language, of what these (Footnote 9–11) identifiers mean.

3.1. Formal Specification of Specific Domains

Once we have decided to describe a specific domain then we can avail ourselves of using one or more of a set of formal specification languages. But such a formal specification does not give meaning to identifiers of the categories and predicate and observer functions; they give meaning to very specific subsets of such categories and predicate and observer functions. And the domain specification now ascribes, not the real thing, but usually some form of mathematical structures as models of the specified domain.

3.2. Formal Domain Specification Languages

There are, today, a large number of formal specification languages. Some are textual, some are diagrammatic (or graphic). The textual specification languages are like mathematical expressions, that is: linear text, often couched in an abstract "programming language" notation. The diagrammatic specification languages provide for the specifier to draw two-dimensional figures composed from primitives. Both forms of specification languages have precise mathematical meanings, but the linear textual ones additionally provide for proof rules.

Examples of textual, formal specification languages are

- Alloy [33]: model-oriented,
- B, Event-B [1]: model-oriented,
- CafeOBJ [19]: property-oriented (algebraic),
- CASL [16]: property-oriented (algebraic),
- CSP [30]: communicating sequential processes,
- DC (Duration Calculus) [54]: temporal logic,
- Maude [14, 38, 12]: property-oriented (algebraic),
- RAISE, RSL [21]: property and model-oriented,
- TLA+ [34]: temporal logic and sets,

[10]Some predicates: is_ENTITY, is_SIMPLE_ENTITY, is_ACTION, is_EVENT, is_BEHAVIOUR, is_ATOMIC, is_COMPOSITE, is_DISCRETE_SIMPLE_ENTITY, is_DISCRETE_SIMPLE_ENTITY, is_DISCRETE_SIMPLE_ENTITY, is_CONTINUOUS_SIMPLE_ENTITY, is_SET, is_CARTESIAN, is_LIST, is_MAP, is_GRAPH, etc.

[11]Some observers: obs_SIMPLE_ENTITY, obs_ACTION, obs_EVENT, obs_BEHAVIOUR, obs_ATTRIBUTE, obs_NAME, obs_TYPE, obs_VALUE, obs_SET, obs_CARTESIAN, obs_ARITY, obs_LIST, obs_LENGTH, obs_DEFINITION_SET, obs_RANGE, obs_IMAGE, obs_GRAPH, obs_PRECEDING_SIMPLE_ENTITIES, obs_SUCCEEDING_SIMPLE_ENTITIES, obs_MEREOLOGY, obs_INPUT_STATE, obs_ARGUMENT, obs_RESULT_STATE, obs_STIMULUS, obs_EVENT_PREDICATE, obs_BEFORE_STATE, obs_AFTER_STATE, etc.

• VDM, VDM-SL [18]: model-oriented and
• Z [53]: model-oriented.

DC and TLA+ are often used in connection with either a model-oriented specification languages or just plain old discrete mathematics notation !

But the model-oriented specification languages mentioned above do not succinctly express concurrency, but CSP does. The diagrammatic, formal specification languages, listed below, all do that:

• Petri Nets [47],
• Message Sequence Charts (MSC) [32],
• Live Sequence Charts (LSC) [29] and
• Statecharts [28].

4. CONCLUSION

4.1. What Have We Done ?

We have discussed a number of meta-linguistic constructs, each corresponding to a description concept: *entities* as *simple entities*, *actions*, *events* and *behaviours*; `categories`, i.e., types, `is_` predicates, `obs_`erver functions and `meta-conditions`, i.e., axioms.

We have used these meta-linguistic constructs in an inquiry into what one may wish, or be able, to describe. Throughout we have not kept these meta-linguistic constructs far from our “ordinary” means of formal description – as illustrated in the large example of Sec. 2 of part I of this paper [3].

4.2. What Need to Be Done?

What needs to be done is more work on the specification ontology, that is, the one promulgated in Sec. 2 of this paper. Work in the direction of establishing a reasonably comprehensice axiom system for this specification ontology and proving properties of this ontology. In a separate paper [D. Bjørner. UNU-IIST: Science $\rightarrow$ Engineering $\rightarrow$ Technology], which is to be submitted for publication, we are covering a triplet of concepts: laws of domain descriptions, laws of domain models, and laws of domains. We think that having a reasonably well-founded specification ontology will allows us to prove properties such as some of these laws.

4.3. Acknowledgements

I think that a first treatment of some issues of computing science versus some possible philosophical issues took place during one of Prof. Nikolaj Nikitchenko’s visits to the Technical University of Denmark in the 1980s and during a fall of 1996 visit to UNU-IIST, Macau. We were experimenting with models of transportation. Nikolaj then brought up some work of Georg Wilhelm Friedrich Hegel and, alone and with my colleague Bo Stig Hansen wrote, over the years some reports and published some papers [39, 40, 26, 27]. The work of this paper tries a slightly different approach. Perhaps it is time to review this work in the context of [39, 40, 26, 27]. Anyway I thank Nikolaj for his visits to Denmark and Macau where, in both places, he gave wonderful, thoughtful seminars and talks.

I am very grateful to Tony Hoare for renewing my interest in the works of Bertrand Russell. I am also grateful to Stephen Linton of the University of St. Andrews, Scotland for inviting me to present a paper from which the present one is quite a revision. I am likewise grateful to Alan Bundy of the University of Edinburgh, Scotland, for a six week distinguished CISA guest Researcher invitation where I had time to write the first version of this paper.

Finally I am grateful to Profs. Alexander Letichevsky and Nikolaj Nikitchenko of Glushkov Institute of Cybernetics, Institute of Program Systems, for inviting me to this workshop and to Ukraine. And I am deeply grateful to Mr. Ievgen Ivanov for his work on making this paper ready for publication.

In response to my paper on *Mereologies in Computing Science* [7] for Tony Hoare's 75 anniversary Festschrift April 2009 (Springer Series on Hisotry of Computing) where I focused on mereologies and a relation between mereologies and CSP [30], Tony brought up, in private communication, Bertrand Russell's concept of *Logical Atomism* [49].

Some of the mereology aspects of this paper have been dealt with in more detail in [10, 7]; while [6] covered other aspects, in an early form.

REFERENCES

1. J.-R. Abrial. The B Book: Assigning Programs to Meanings and Modeling in Event-B: System and Software Engineering. Cambridge University Press, Cambridge, England, 1996 and 2009.
2. P. Bernus and L. Nemes, editors. Modelling and Methodologies for Enterprise Integration, International Federation for Information Processing, London, UK, 1996 1995. IFIP TC5, Chapman & Hall. Working Conference, Queensland, Australia, November 1995.
3. D. Bjørner. Domain Science and Engineering. From Computer Science to The Sciences of Informatics. Part I of II: Engineering // Kibernetika i Systemni Analiz. — 2010. — **46**, N 4. — P. 100–116.
4. D. Bjørner. Software Engineering, Vol. 3: Domains, Requirements and Software Design. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
5. D. Bjørner. Domain Theory: Practice and Theories, Discussion of Possible Research Topics. In ICTAC'2007, volume 4701 of Lecture Notes in Computer Science (eds. J.C.P. Woodcock et al.), pages 1–17, Heidelberg, September 2007. Springer. Final Version.
6. D. Bjørner. An Emerging Domain Science – A Role for Stanislaw Lesniewski's Mereology and Bertrand Russell's Philosophy of Logical Atomism. Higher-order and Symbolic Computation, 2009. Final Version.
7. D. Bjørner. On Mereologies in Computing Science. In Festschrift for Tony Hoare, History of Computing (ed. Bill Roscoe), London, UK, 2009. Springer. Final Version.
8. D. Bjørner. Domain Engineering. In BCS FACS Seminars, Lecture Notes in Computer Science, the BCS FAC Series (eds. Paul Boca and Jonathan Bowen), pages 1–42, London, UK, 2010. Springer. Final Version.
9. D. Bjørner. Domain Engineering: Technology Management, Research and Engineering. JAIST Press, March 2009. JAIST Research Monograph #4, 536 pages.
10. D. Bjørner and A. Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness, volume 5930 of Lecture Notes in Computer Science, pages 22–59, Heidelberg, July 2010. Springer. Final Version.
11. W.D. Blizard. A Formal Theory of Objects, Space and Time. The Journal of Symbolic Logic, **55**(1):74–89, March 1990.
12. R. Bruni and J. Meseguer. Generalized Rewrite Theories. In Jos C. M. Baeten and Jan Karel Lenstra and Joachim Parrow and Gerhard J. Woeginger, editor, Automata, Languages and Programming. 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30–July 4, 2003. Proceedings, volume 2719 of Lecture Notes in Computer Science, pages 252–266. Springer-Verlag, 2003.
13. W. Clancey. The knowledge–level reinterpreted: modeling socio–technical systems. International Journal of Intelligent Systems, 8:33–49, 1993.

14. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. The Maude 2.0 System. In Robert Nieuwenhuis, editor, Rewriting Techniques and Applications (RTA 2003), number 2706 in Lecture Notes in Computer Science, pages 76–87. Springer-Verlag, June 2003.

15. N. Cocchiarella. Formal Ontology. In H. Burkhardt and B. Smith, editors, Handbook in Metaphysics and Ontology, pages 640–647. Philosophia Verlag, Munich, Germany, 1991.

16. CoFI (The Common Framework Initiative). Casl Reference Manual, volume 2960 of Lecture Notes in Computer Science (IFIP Series). Springer–Verlag, 2004.

17. D.J. Farmer. Being in time: The nature of time in light of McTaggart's paradox. University Press of America, Lanham, Maryland, 1990. 223 pages.

18. J. Fitzgerald and P. G. Larsen. Modelling Systems – Practical Tools and Techniques in Software Development. Cambridge University Press, Cambridge, UK, Second edition, 2009.

19. K. Futatsugi, A. Nakagawa, and T. Tamai, editors. CAFE: An Industrial–Strength Algebraic Formal Method, Sara Burgerhartstraat 25, P.O. Box 211, NL–1000 AE Amsterdam, The Netherlands, 2000. Elsevier. Proceedings from an April 1998 Symposium, Numazu, Japan.

20. B.H.P. Gennaro Chierchia and R. Turner, editors. Properties, Types and Meaning. Kluwer Academic, 15 December 1988. Vol. I: Foundational Issues, Vol. II: Semantic Issues.

21. C.W. George, A.E. Haxthausen, S. Hughes, R. Milne, S. Prehn, and J.S. Pedersen. The RAISE Development Method. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

22. T.R. Gruber and G.R. Olsen. An Ontology for Engineering Mathematics. In J. Doyle, P. Torasso, and E. Sandewall, editors, Principles of Knowledge Representation and Reasoning. Morgan Kaufmann, 1994. Fourth International Conference. Gustav Stresemann Institut, Bonn, Germany. http://www-ksl.stanford.edu/knowledge-sharing/papers/engmath.html

23. M. Gruninger and M. Fox. The Logic of Enterprise Modelling. In Modelling and Methodologies for Enterprise Integration, pages 141–157, November 1995.

24. N. Guarino. Formal Ontology, Conceptual Analysis and Knowledge Representation. Intl. Journal of Human–Computer Studies, 43:625–640, 1995.

25. N. Guarino. Some Organising Principles for a Unifed Top–level Ontology. Int. rept., Italian National Research Council (CNR), LADSEB–CNR, Corso Stati Uniti 4, I–35127 Padova, Italy , 1997.

26. B. Hansen and N. Nikitchenko. Abstract transport systems: An initial study. Research Report IT-Dok: 1998-006, Technical University of Denmark, 1998.

27. B. Hansen and N. Nikitchenko. Abstract Transport Systems: Compositions and Description Languages. In Proc. of the First Int. Conf. on Programming UkrProg98, Ukraine, Kiev, 1998.

28. D. Harel. Statecharts: A visual formalism for complex systems. Science of Computer Programming, 8(3):231–274, 1987.

29. D. Harel and R. Marelly. Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine. Springer-Verlag, 2003.

30. C. Hoare. Communicating Sequential Processes. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985. Published electronically: http://www.usingcsp.com/-cspbook.pdf (2004).

31. C. Hoare and D. Allison. Incomputability. ACM Comput. Surv., **4**(3):169–178, 1972.

32. ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992, 1996, 1999.

33. D. Jackson. Software Abstractions: Logic, Language, and Analysis. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

34. L. Lamport. Specifying Systems. Addison–Wesley, Boston, Mass., USA, 2002.

35. J. McCarthy. Towards a Mathematical Science of Computation. In C. Popplewell, editor, IFIP World Congress Proceedings, pages 21–28, 1962.

36. J.M.E. McTaggart. The Unreality of Time. Mind, **18**(68):457–84, October 1908. New Series.

37. D.H. Mellor and A. Oliver. Properties. Oxford Readings in Philosophy. Oxford Univ Press, May 1997. ISBN: 0198751761, 320 pages.

38. J. Meseguer. Software Specifcation and Verification in Rewriting Logic. NATO Advanced Study Institute, 2003.

39. N. Nikitchenko. Towards foundations of the general theory of transport domains. Research Report 88, UNU/IIST, 1996.

40. N. Nikitchenko. Construction of abstract transport models oriented on composition programming systems. Problems of Programming (In Russian), (2), 1997.

41. R.L. Poidevin and M. MacBeath, editors. The Philosophy of Time. Oxford University Press, 1993.

42. A. Prior. Changes in Events and Changes in Things. Oxford University Press, 1993.

43. A.N. Prior. Logic and the Basis of Ethics. Clarendon Press, Oxford, UK, 1949.

44. A.N. Prior. Time and Modality. Oxford University Press, Oxford, UK, 1957.

45. A.N. Prior. Past, Present and Future. Clarendon Press, Oxford, UK, 1967.

46. A.N. Prior. Papers on Time and Tense. Clarendon Press, Oxford, UK, 1968.

47. W. Reisig. Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien. Institut für Informatik, Humboldt Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany, 1 Oktober 2009. 276 pages. http://www2.informatik.hu-berlin.de/top/pnene buch/pnene buch.pdf.

48. G. Rochelle. Behind time: The incoherence of time and McTaggart's atemporal replacement. Avebury series in philosophy. Ashgate, Brookfield, Vt., USA, 1998. vii + 221 pages.

49. B. Russell. The Philosophy of Logical Atomism. The Monist: An International Quarterly Journal of General Philosophical Inquiry, xxxviii–xxxix:495–527, 32–63, 190–222, 345–380, 1918–1919.

50. J.F. Sowa. Knowledge Representation: Logical, Philosophical, and Computational Foundations. Pws Pub Co, August 17, 1999. ISBN: 0534949657, 512 pages.

51. S. Staab and R. Stuber, editors. Handbook on Ontologies. International Handbooks on Information Systems. Springer, Heidelberg, 2004.

52. J. van Benthem. The Logic of Time, volume 156 of Synthese Library: Studies in Epistemology, Logic, Methhodology, and Philosophy of Science (Editor: Jaakko Hintika). Kluwer Academic Publishers, P.O.Box 17, NL 3300 AA Dordrecht, The Netherlands, second edition, 1983, 1991.

53. J.C.P. Woodcock and J. Davies. Using Z: Specification, Proof and Refinement. Prentice Hall International Series in Computer Science, 1996.

54. C.C. Zhou and M.R. Hansen. Duration Calculus: A Formal Approach to Real–time Systems. Monographs in Theoretical Computer Science. An EATCS Series. Springer–Verlag, 2004.

Поступила 09.04.2010