Domain Science & Engineering
From Computer Science to The Sciences of Informatics

Dines Bjørner
Fredsvej 11, DK-2840 Holte, Denmark
bjorner@gmail.com -- www.imm.dtu.dk/~db

14 February 2010: Compiled: March 13, 2010: 00:00 ECT

Abstract

In this paper we wish to **advocate** (i) that schools, institutes and departments of computer science, software engineering, informatics, cybernetics, and the like, **re-orient** themselves along two lines: (i.1) more **emphasis on teaching** programming and software engineering based on **formal methods**; and (i.2) more **emphasis on research** into **formal methods** for the trustworthy development of software that meets customers' expectations and is correct, that is, the right software and that the software is right. We also wish to **advocate** (ii) that the concepts of **domain science** and **domain engineering** become an indispensable part of the **science of informatics** and of **software engineering**. And we finally wish to **advocate** (iii) that informatics research centers embark on **path-finder projects** which **research** and **experimentally develop** domain models for infra-structure components, for example, (iii.1) **financial service industries** (banks, stock exchanges, etc.), (iii.2) **health-care** (hospitals, clinics, private physicians, etc.) (iii.3) **pipeline systems** (oil, gas), (iii.4) **transportation** (such as railways, shipping, air traffic, etc.).

Contents

1 Introduction 5

The background postulates of this paper are the following: (i) half a century of computer science research may very well have improved our understanding of computing devices (automata etc.), but it has yet to contribute significantly to the quality of software products; (ii) our students, the future leading software engineers, those of them who go into industry rather than "remaining" in academia, are being mislead by too many foundational courses

to believe that these are relevant for the practice of software engineering; (iii) a significant re-orientation of university teaching and research into both 'computer science' and software engineering must occur if we are to improve the relevance of 'computer science' to software engineering. In this paper we shall, unabashedly, suggest the kind of re-orientation that we think will rectify the situation alluded to in Items (i–iii). 6

1.1 Some Definitions of Informatics Topics 7

Let us first delineate our field of study. It first focuses on *computer science*, *computing science*, *software* and *software engineering*. 8

Definition 1 – ***Computer Science:*** By *computer science* we shall understand the study and knowledge of the properties of the *'things'* that can *'exist'* inside computers: data and processes.

Examples of *computer science* disciplines are: automata theory (studying automata [finite or otherwise] and state machines [without or with stacks]), formal languages (studying, mostly the syntactic the "foundations" and "recognisability" of abstractions of of computer programming and other "such" languages), complexity theory, type theory, etc. 9

Some may take exception to the term *'things'*[1] used in the above and below definition. They will say that it is imprecise. That using the germ conjures some form of reliance on Plato's Idealism, on his Theory of Forms. That is, *"that it is of Platonic style, and thus, is disputable. One could avoid this by saying that these definitions are just informal rough explanations of the field of study and further considerations will lead to more exact definitions."*[2] Well, it may be so. It is at least a conscious attempt, from this very beginning, to call into dispute and discuss "those things". Section 2 of this paper ("A Specification Ontology and Epistemology") has as one of its purposes to encircle the problem. 10

Definition 2 – ***Computing Science:*** By *computing science* we shall understand the study and knowledge of the how to construct the *'things'* that can *'exist'* inside computers: the software and its data.

Conventional examples of *computing science* disciplines are: algorithm design, imperative programming, functional programming, logic programming, parallel programming, etc. To these we shall add a few in this paper. 11

Definition 3 – ***Software:*** By *software* we shall understand not only the code intended for computer execution, but also its use, i.e., programmer manuals: installation, education, user and other guidance documents, as well all as its development documents: domain models, requirements models, software designs, tests suites, etc. "zillions upon zillions" of documents. 12

The fragment description of the example Pipeline System of this paper exhibits, but a tiny part of a domain model.

[1] and also to the term *'exist'*.

[2] Cf. personal communication, 12 Feb., 2010, with Prof. Mikula Nikitchenko, Head of the Chair of Programming Theory of Shevchenko Kyiv National University, Ukraine

Definition 4 – ***Software Engineering:*** By *software engineering* we shall understand the methods (analysis and construction principles, techniques and tools) needed to carry out, manage and evaluate software development projects as well as software product marketing, sales and service — whether these includes only domain engineering, or requirements engineering, or software design, or the first two, the last two or all three of these phases. *Software engineering*, besides documents for all of the above, also includes all auxiliary project information, stakeholder notes, acquisition units, analysis, terminology, verification, model-checking, testing, etc. documents

1.2 The Triptych Dogma 13

Dogma 1 – ***Triptych:*** By *the triptych dogma* we shall understand a dogma which insists on the following: Before software can be designed one must have a robust understanding of its requirements; and before requirements can be prescribed one must have a robust understanding of their domain.

14

Dogma 2 – ***Triptych Development:*** By *triptych development* we shall understand a software development process which starts with one or more stages of *domain engineering* whose objective it is to construct a *domain description*, which proceeds to one or more stages of *requirements engineering* whose objective it is to construct a *requirements prescription*, and which ends with one or more stages of *software design* whose aim it is to construct the *software*.

1.3 Structure of This Paper 15

In Sect. **??** we present a non-trivial example. It shall serve to illustrate the new concepts of *domain engineering*, *domain description* and *domain model*. In Sect. **??** we shall then discuss ramifications of the triptych dogma. Then we shall follow-up, Sect. 2, on what we have advocated above, namely a beginning discussion of our logical and linguistic means for description, of "the kind of *'things'* that can *'exists'* or the things (say in the domain, i.e., "real world") that they reflect".

2 A Specification Ontology and Epistemology[3]

2.1 A Twofold Problem

The twofold problem is the following. (1) There is a dichotomy between what an informal description, a narrative, and what a formal description, a formalisation, refers to. (2) And which are our means of description? 16

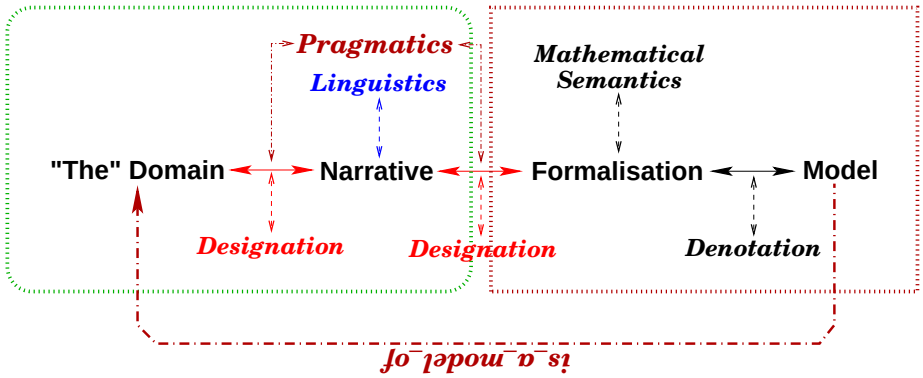

Figure 1: Formal and Informal Relations

Problem (1) is indicated in Fig. 1. 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 clear a clear, didactic understanding of the domain, and is pedagogical, that is, introduces domain phenomena and notions, gently, one-by-one. (1.1) In Sect. ?? on page ?? we presented a pair of descriptions 17
— 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. (1.2) In the formal part of the 18
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, in (1.1–1.2), lies the first 19

[3]Webster Collegiate Dictionary explanation of philosophical terms:

(i–ii) 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.

(iii) Epistemology: the study or a theory of the nature and grounds of knowledge especially with reference to its limits and validity

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!"*[4] One can not establish a formal relation between the informal world of domains and the formal world of mathematical specifications. **The** 20 **above is an essence of abstract models.**

(2.) Then we outline the "describability" problem. **What, of actual domains, can be modelled ?** That is, what, of a[ny] 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 on the preceding page.

2.1.1 Russell's Problem 21

Bertrand Russell, in [**?**] 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 → DESIGNATION-**infset**
  designation: Sentence → DOMAIN $\xrightarrow{\sim}$ DESIGNATION
  existance: Sentence → DOMANI → **Bool**
  designation(s)(ω) ≡ **if** existance(s)(ω) **then** ... **else chaos end**

I claim that Russell's "problem" lies in the **green** dashed [rounded edge] box of the left side of Fig. 1 on the previous page. The simpler-minded computer/computing science problem of syntax and semantics is then that of (the **brown** rectangular box of) the right side of Fig. 1 on the preceding page.

2.1.2 The Problem of This Paper 22

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 on the previous page: 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).

[4] Rudyard Kipling, *Barrack-room Ballads*, 1892: *"Oh, East is East, and West is West, and never the twain shall meet."*

2.2 What Is An Ontology ? 23

2.2.1 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 Sects. 2.3–2.6 and 2.8–2.9. The pragmatics of description designations 24
(using these syntactic constructs) are, of course, phenomena of the domain. The semantics of description designations are, of course, mathematical constructs. Thus we have a duality here: On one hand we have that the pragmatics, that is, the intention of our use of descriptions, is that they designate phenomena of an actual world, whereas, on the other hand, the semantics, that is, the formal meaning of descriptions, is that they denote mathematical concepts.

Is there a problem here ?

And: what can we describe ? 25

Yes, there is a "slight problem"? We cannot, in fact never, establish a formal relationship between the pragmatics and the semantics.

And: On one hand there is a world, that is, a set of domains, to be described. And, on the other hand, there are some syntactic constructs that can be used in providing such descriptions. Are the latter sufficient to describe all that we wish to describe? Well, we shall never know. So it is a conjecture, not a theory of description ontologies. 26

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.

2.2.2 Description Ontology Versus Ontology Description 27

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 **??** emphasized the need for describing domain phenomena and concepts. This section 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. 28

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' [**?**, **?**, **?**, **?**, **?**, **?**, **?**, **?**].

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.

2.3 Categories, Predicates and Observers for Describing Domains 29

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 [?, ?, ?, ?, ?]. 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 [?, John Sowa, 1999] than the one put forward in this paper.

2.3.1 An Aside on Notation 30

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.

2.3.2 The Hypothetical Nature of Categories, Predicates and Observers 31

In the following we shall postulate some categories of phenomena[5], that is, some meta-types:

categories
𝔸𝕃ℙ𝕀𝔸, 𝔹𝔼𝕋𝔸, 𝔾𝔸𝕄𝕄𝔸

What such a clause as the above means is that we postulate that there are the categories 𝔸𝕃ℙ𝕀𝔸, 𝔹𝔼𝕋𝔸, 𝔾𝔸𝕄𝕄𝔸 of "things" (phenomena and concepts) in the world of domains.
32 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" !

2.3.3 Predicates and Observers 33

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

[5]These observable phenomena or abstract concepts are, in general, entities, more specifically either simple entities, actions, events or behaviours.

predicate signatures
is_ALPHA: "Things" → **Bool**
is_BETA: "Things" → **Bool**
is_GAMMA: "Things" → **Bool**
observer signatures
obs_ALPHA: "Things" $\xrightarrow{\sim}$ ALPHA
obs_BETA: ALPHA $\xrightarrow{\sim}$ BETA
obs_GAMMA: ALPHA $\xrightarrow{\sim}$ GAMMA

So we are "fixing" a logic ! 34

The "Things" clause is a reference to the domain under scrutiny. Some 'things' in that domain are of category[6] 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.

2.3.4 Uncertainty 35

The function signature:

value
fct: A $\xrightarrow{\Re}$B

expresses a relation, $\mathsf{fct}_{\Re}$, which one may think of as:

type
$\mathrm{FCT}_{\Re}$ = (A × B)**-infset**.

Applying fct to an argument a, that is, f(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: $\mathsf{a} \notin \mathsf{fct}_{\Re}$; or sometimes result in some b, sometimes in another b′, etc., that is, $\mathsf{fct}_{\Re}=\{(\mathsf{a},\mathsf{b}), (\mathsf{a},\mathsf{b}'), (\mathsf{a},\mathsf{b}''), \ldots\}$.

2.3.5 Meta-Conditions 36

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.

[6] 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.

2.3.6 Discussion 37

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, ..., is_A, is_B, ... obs_A, obs_B, ... **meta-condition [axiom]** predicate notions derive from McCarthy's *analytic syntax* [**?**].

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 **categories, 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.

2.4 Entities 38

What we shall describe is what we shall refer to as entities. In other words, there is a category and meta-logical predicate 𝔼𝕟𝕥𝕚𝕥𝕪, 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" → **Bool**
meta condition:
∀ 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. 39

• • •

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.

2.4.1 Entity Categories 40

We postulate four entity categories:

category:
SIMPLE_ENTITY, ACTION, EVENT, BEHAVIOUR

Some **phenomena** or **concepts** are simple entities. **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. 41

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. 42

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. 43

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. 44

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" $\overset{\Re}{\rightarrow}$ **Bool**
is_ACTION "Thing" $\overset{\Re}{\rightarrow}$ **Bool**
is_EVENT "Thing" $\overset{\Re}{\rightarrow}$ **Bool**
is_BEHAVIOUR "Thing" $\overset{\Re}{\rightarrow}$ **Bool**

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

is_ENTITY(t)

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

The $\cup$ "union" is inclusive:

meta condition:

$\forall$ t:``Thing''•is_ENTITY(t) $\Rightarrow$
is_SIMPLE_ENTITY(t) $\vee$ is_ACTION(t) $\vee$ is_EVENT(t) $\vee$ is_BEHAVIOUR(t)

2.5 Simple Entities

We postulate that there are **atomic** simple entities, that there are [therefrom distinct] **composite** simple entities, and 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 $\cup$ COMPOSITE

observer signature:

is_ATOMIC: SIMPLE_ENTITY $\rightarrow$ **Bool**
is_COMPOSITE: SIMPLE_ENTITY $\rightarrow$ **Bool**

meta condition:

ATOMIC $\cap$ COMPOSITE = {}
$\forall$ s: "Things":SIMPLE_ENTITY •
is_ATOMIC(s) $\equiv \sim$is_COMPOSITE(s)

Discussion: We put in brackets, in the text paragraph before the above formulas, [therefrom 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' [**?**].

2.5.1 Discrete and Continuous Entities 48

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 ∪ CONTINUOUS_SIMPLE_ENTITY
predicate signatures:
is_DISCRETE_SIMPLE_ENTITY: SIMPLE_ENTITY → **Bool**
is_CONTINUOUS_SIMPLE_ENTITY: SIMPLE_ENTITY → **Bool**
meta condition:
[is it desirable to impose the following]
∀ s:SIMPLE_ENTITY •
is_DISCRETE_SIMPLE_ENTITY(s) ≡ ∼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.

2.5.2 Attributes 49

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, NAME, TYPE, VALUE
observer signature:
obs_ATTRIBUTEs: SIMPLE_ENTITY → ATTRIBUTE-**set**
obs_NAME: ATTRIBUTE → NAME
obs_TYPE: ATTRIBUTE → TYPE
obs_VALUE: ATTRIBUTE → VALUE
meta condition:
∀ s:SIMPLE_ENTITY •
∀ a,a′:ATTRIBUTE • {a,a′}⊆obs_ATTRIBUTEs(s)
∧ a≠a′ ⇒ obs_NAME(a)≠obs_NAME(a′)

50 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`
51 `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 [?, Properties] and [?, Properties, Types and Meaning] for philosophical discourses on attributes.

2.5.3 Atomic Simple Entities: Attributes

52

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'.

2.5.4 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

53

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 → SIMPLE_ENTITY-**set**

Mereology

54

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 → **Bool**

55

- **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 → **Bool**
observer signatures:
obs_ARITY: COMPOSITE $\xrightarrow{\sim}$ **Nat**
pre: obs_ARITY(s): is_CARTESIAN(s)
obs_CARTESIAN: COMPOSITE $\xrightarrow{\sim}$
SIMPLE_ENTITY × ... × SIMPLE_ENTITY
pre obs_CARTESIAN(s): is_CARTESIAN(s)

56

meta condition:
∀ c:SIMPLE_ENTITY•
is_COMPOSITE(c)∧is_CARTESIAN(c) ⇒
obs_SIMPLE_ENTITIES(c) = **elements of** obs_CARTESIAN(c)
∧ **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 distinct(ly 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.

57

- **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)

58

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:
∀ c:SIMPLE_ENTITY • is_COMPOSITE(c) ∧ is_GRAPH(c)
⇒ **let** ss = SIMPLE_ENTITIES(c) **in**
∀ s′:SIMPLE_ENTITY • s′ ∈ ss
⇒ obs_PRECEEDING_SIMPLE_ENTITIES(c)(s′) ⊆ ss

$\wedge$ obs_SUCCEEDING_SIMPLE_ENTITIES(c)(s′) $\subseteq$ ss
end

61

2.5.5 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), etcetera. predicates and using also the observer functions sketched above. 62

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 summery: If ATOMIC then ATTRIBUTES can be analysed. 63
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.

2.5.6 Practice 64

How do we interpret this section, Sect. 2.5, on simple entities? We practice it by analysing the domain according to the principles laid down in this section, Sect. 2.5, 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, 65
COMPOSITE, DISCRETE_SIMPLE_ENTITY, CONTINUOUS_SIMPLE_ENTITY, ATTRIBUTE, NAME, TYPE, VALUE, CARTESIAN, ARITY, LIST, LENGTH, GRAPH, PRECEDING_SIMPLE_E SUCCEEDING_SIMPLE_ENTITIES, etc., nor their is_ or obs_ forms. The example of Sect. ?? has given several examples of sort and type definitions as well as of constraint definitions.

2.6 Actions 66

2.6.1 Definition

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.

2.6.2 Non-Observables 67

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 are purely a mathematical construction, possessing a number of properties and introduced here in order to explain what is going on in a(ny) domain. We shall not be able to observe functions. To highlight this point we use this *SPELLING OF FUNCTIONS*.

2.6.3 Observers &c. 68

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

category:
ACTION, *FUNCTION*, STATE
predicate signature:
is_ACTION: ENTITY → **Bool**

69

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. Not all elements
70 of the CARTESIAN ARGUMENT are SIMPLE STATE ENTITIES.

category:
STATE = SIMPLE_ENTITY
FUNCTION = SIMPLE_ENTITY × STATE → STATE
ARGUMENT = {|s:SIMPLE_ENTITY•is_CARTESIAN(s)|}
observer signatures:
obs_ACTION: ENTITY → ACTION
obs_ARGUMENT: ACTION → ARGUMENT
obs_INPUT_STATE: ACTION → STATE
obs_RESULT_STATE: ACTION → STATE

2.6.4 Practice 71

How do we interpret this section, Sect. 2.6, on actions? We practice it by analysing the domain according to the principles laid down in this section, Sect. 2.6, 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 their is_ or obs_ forms. The earlier example of Sect. ?? has given several examples of action signatures and definitions.

2.7 "Half-way" Discussion[7] 72

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 observe 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 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 does not necessarily represent the "truth". They hopefully are not "falsities". At best they are abstractions and, as such, they are approximations.

2.8 Events 73

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

2.8.1 Definition of Atomic Events 74

By an EVENT we shall understand A pair, (σ, σ'), of STATEs, a STIMULUS, s, (which is like a *FUNCTION* of an ACTION), and an EVENT PREDICATE, $p : \mathcal{P}$, such that $p(\sigma, \sigma')(s)$, 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.

2.8.2 Examples of Atomic Events 75

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.

[7] "Halfway": after simple entities and actions and before events and behaviours.

2.8.3 Composite Events 76

Definition

A Composite event is composed from a sequence of two or more events: The 'after' state of one event becomes the 'before' event of the immediately subsequent event We

Examples of Composite Events 77

2.8.4 Observers &c. 78

We postulate that the domain engineer from an EVENT can observe the STIMULUS, the BEFORE_STATE, the AFTER_STATE and the EVENT_PREDICATE. As said before:
79 the domain engineer cum researcher can decide on these abstractions, these approximations.

category:
  STIMULUS = SIMPLE_ENTITY × STATE → STATE
  $\mathcal{P}$ = STATE × STATE → **Bool**
observer signatures:
  obs_STIMULUS: EVENT → STIMULUS
  obs_BEFORE_STATE: EVENT → STATE
  obs_AFTER_STATE: EVENT → STATE
  obs_EVENT_PREDICATE: EVENT → $\mathcal{P}$
meta condition:
  ∀ e:EVENT •
    ∃ s:STIMULUS •
      INPUT_STATE(e) = BEFORE_STATE(s) ∧
      RESULT_STATE(e) = AFTER_STATE(s) ∧
      ∃ p:$\mathcal{P}$ •p(s)(INPUT_STATE(e),RESULT_STATE(e))

2.8.5 Practice 80

How do we interpret this section, Sect. 2.8, on events? We practice it by analysing the domain according to the principles laid down in this section, Sect. 2.8, 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)
81 and writing down their definitions (as if they were domain actions). 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. The example of Sect. ?? has not given very many examples.

So we give some narrative examples, that is, without formalisations: 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.

2.9 Behaviours 82

2.9.1 A Loose Characterisation

By a BEHAVIOUR we shall understand a set of sequences of ACTIONs and EVENTs one sequence of which designates the behaviour of the environment the remaining sequences designate behaviours of different "overlapping" or "disjoint" parts of the domain. It may now be so that some EVENTs in two or more such sequences have their STATEs 83
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.

2.9.2 Observers &c. 84

We abstract from the orderly example of synchronisation and communication given above and introduce a further un-explained notion of behaviour (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. 85

We can classify some BEHAVIOURs.

(i) SIMPLE SEQUENTIAL BEHAVIOURs are sequences of ACTIONs.

(ii) SIMPLE_CONCURRENT_BEHAVIOURs are sets of SIMPLE_SEQUENTIAL_BEHAVIOURs.

(iii) COMMUNICATING_CONCURRENT_BEHAVIOURs are sets of sequences of ACTIONs and BEHAVIOUR_INTERACTION_LABELs. 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. 86

Many other composite behaviours can be observed. For our purposes it suffice with having just identified the above.

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 87
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` [?]. In addition, more generally, we refer to A.N. Prior's [?, ?, ?, ?, ?] and McTaggart's works [?, ?, ?]. The paper by Wayne D. Blizard [?] proposes an axiom system for time-space.

[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.

2.9.3 Practice 88

How do we interpret this section, Sect. 2.9, on behaviours? We practice it by analysing the domain according to the principles laid down in this section, Sect. 2.9, 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
89 events — the latter modelled as CSP communications between behaviours). 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.

2.10 Mereology 90

Simple entities — when composite — are said to exhibit a mereology. Thus composition of simple entities imply a mereology. We discussed mereologies of behaviours: simple sequential, simple concurrent, communicating concurrent, etc. Above we did not treat actions and events as potentially being composite. But we now relax that seeming constraint. There
91 is, in principle, nothing that prevents actions and events from exhibiting mereologies. An action, still instantaneous, can, for example, "fork" into a number of concurrent actions, all instantaneous, on "disjoint" parts of a state; or an instantaneous action can "dribble" (not little-by-little, but one-after-the-other. still instantaneously) into several actions as if
92 a simple sequential behaviour, but instantaneous. Two or more events can occur simultaneously: two or more (up to four, usually) people become grandparents when a daughter of theirs give birth to their first grandchild; or an event can — again a "dribble" (not little-by-little, but instantaneously) — "rapidly" sequence through a number of instantaneous sub-events (with no intervening time intervals): A bankruptcy events immediately causes the bankruptcy of several enterprises which again causes the immediate bankruptcy
93 of several employes, etcetera.

The problems of compositionality of entities, whether simple, actions, events or behaviours, is was studied, initially, in [?, Bjørner and Eir, 2008]

2.11 Impossibility of Definite Mereological Analysis of Seemingly Composite Entities 94

It would be nice if there was a more-or-less obvious way of "deciphering" the mereology of an entity. In the many • (bulleted) items above (cf. Set, Cartesian, List, Map, Graph) 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. It is a matter of both discovery and choice between seemingly alternative mereologies, and it is also a matter of choice of abstraction.

2.12 What Exists and What Can Be Described ? 95

In the previous section we have suggested a number of *categories*[9] of entities, 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 96

Footnotes 9–11 (Page 23) 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 97

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 98

There are, today, 2010, a large number of formal specification languages. Some or textual, some are diagrammatic. 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. 99

Examples of textual, formal specification languages are

[9] Some categories: ENTITY, SIMPLE_ENTITY, ACTION, EVENT, BEHAVIOUR, ATOMIC, COMPOSITE, DISCRETE, CONTINUOUS, 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.

[10] Some predicates: is_ENTITY, is_SIMPLE_ENTITY, is_ACTION, is_EVENT, is_BEHAVIOUR, is_ATOMIC, is_COMPOSITE, 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.

- Alloy [?]: model-oriented,
- B, Event-B [?]: model-oriented,
- CafeOBJ [?]: property-oriented (algebraic),
- CASL [?]: property-oriented (algebraic),
- CSP [?]: communicating sequential processes,
- DC (Duration Calculus) [?]: temporal logic,
- Maude [?, ?, ?]: property-oriented (algebraic),
- RAISE, RSL [?]: property and model-oriented,
- TLA+ [?]: temporal logic and sets,
- VDM, VDM-SL [?]: model-oriented and
- Z [?]: model-oriented.

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

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

- Petri Nets [?],
- Message Sequence Charts (MSC) [?],
- Live Sequence Charts (LSC) [?] and
- Statecharts [?].

3.3 Discussion: "Take-it-or-leave-it !" 101

With the formal specification languages, not just those listed above, but with any conceivable formal specification language, the issue is: you can basically only describe using that language what it was originally intended to specify, and that, usually, was to specify software ! If, in the real domain you find phenomena or concepts, which it is somewhat clumsy and certainly not very abstract or, for you, outright impossible, to describe, then, well, then you cannot formalise them !

4 Conclusion 102

4.1 What Have We Done ?

4.2 What Need to Be Done ?

4.3 Acknowledgements 103

I am Tony very grateful 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.[12] I am likewise grateful to **Alan Bundy** of the University of Edinburgh, Scotland, for a six week distinguished CISA guest Rresearcher 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. **Evgeni Ivanov** for his work on the translation of this paper into Ukrainian.

5 Bibliographical Notes

In response to my paper on *Mereologies in Computing Science* [?] 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` [?], Tony brought up, in private communication, Bertrand Russell's concept of *Logical Atomism* [?].

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

[12] I expect the present paper to be further revised between its submisson, early march 2010, and its presentation, mid May 2010.