On a Philosophy of Informatics and a Theory of a Science of Informatics — Sketches, Musings & Ruminations[1] s1

Dines Bjørner
The University of Edinburgh[2]

October 2, 2009: 14:12

Abstract s2

This paper is for the academic who is interested in a philosophy of science and a theory of science. The paper proposes some features of a philosophy of informatics and a theory of the science of informatics. We approach some such features by first outlining, in "lay' terms, concepts of computer and computing science, informatics and of a triptych of domains, requirements and software. Domains are a new concept in informatics. We briefly illustrate that domains can be described both informally, that is, using natural (cum national) language, and formally, in mathematics. Then we discuss s3
a number of concepts common to all domains, their simple entities, functions, events and behaviours. We then go on to examine some derivative concepts of these: discrete and continuous as well as atomic and composite entities and the mereology of composite entities, Finally we discuss (i) the problem of description in the context of Bertrand s4
Russell's 'Philosophy of Logical Atomism' and (ii) the problem of describing parts and wholes in the context of Stanisław Leśhniewsky's 'Mereology' and Individuals'; and (iii) a possible set of domain description laws. s5

This is our second attempt at discussing aspects of an emerging philosophy of informatics — some might call it dabbling into such issues. A first attempt was a paper [14] for the journal of *Higher Order and Symbolic Computation*, for a fall 2009 issue in honour of the late Peter Landin. The reader should thus bear with us firstly because of our lack of deeper experience in matters of philosophy, secondly because we only treat a few facets and then only hint at them. s6

We shall therefore be grateful for any suggestions from readers, and we are likewise grateful to *The University of St Andrews* for asking me to compose my thoughts on this issue.

Contents

1 Introduction **3**
1.1 Informatics . . . 3
1.1.1 Computer Science . . . 4
1.1.2 Computing Science . . . 4
1.1.3 Software Engineering . . . 4
1.1.4 IT: Information Technology . . . 4
1.1.5 Applications . . . 5
1.2 Structure of Paper . . . 5

2 Software Engineering **6**
2.1 Paraphrasing 'Software Engineering' . . . 6
2.2 Documents – Documents – Documents . . . 6
2.3 Domain Engineering . . . 6
2.3.1 Domain Acquisition: Domain Models and Their Acceptance . . . 7
2.3.2 Domains: Descriptions and Models . . . 7
Domain Intrinsics . . . 7

[1] Paper prepared for The School Colloquium, The University of St Andrews, Tuesday 6 October 2009
[2] SICSA Distinguished Visitor at The University of Edinburgh, Sept. 21 – Oct. 30, 3009
Fredsvej 11, DK-2840 Holte, Denmark. bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Domain Support Technologies . . . 8
Domain Management and Organisation . . . 8
Domain Rules and Regulations . . . 8
Domain Scripts: Licenses and Contracts . . . 9
Domain Human Behaviour . . . 9
2.3.3 Discussion . . . 9

3 A Description Ontology 9
3.1 Description Ontology Versus Ontology Description . . . 9
3.2 Categories, Predicates and Observers for Describing Domains . . . 10
3.2.1 The Hypothetical Nature of Categories, Predicates and Observers . . . 10
Categories . . . 10
Predicates and Observers . . . 10
Meta-Conditions . . . 10
Discussion . . . 11
3.2.2 Entities . . . 11
3.2.3 Entity Categories . . . 12
3.2.4 Simple Entities . . . 12
Discrete and Continuous Entities . . . 13
Attributes . . . 13
Atomic Simple Entities: Attributes . . . 14
Composite Simple Entities: Attributes, Sub-entities and Mereology . . . 14
Sub-entities . . . 14
Mereology (I) . . . 14
Set Mereology . . . 14
Cartesian Mereology . . . 14
List Mereology . . . 15
Map Mereology . . . 15
Graph Mereology . . . 16
3.2.5 Actions . . . 16
3.2.6 Events . . . 17
3.2.7 Behaviours . . . 18
3.2.8 Mereology (II) . . . 19
Compositionality of Entities . . . 19
Impossibility of Definite Mereological Analysis of Seemingly Composite Entities . . . 19
3.3 What Exists and What Can Be Described ? . . . 19
3.3.1 Description Versus Specification Languages . . . 19
Formal Specification of Specific Domains . . . 20
Formal Domain Specification Languages . . . 20
Discussion: “Take-it-or-leave-it !” . . . 20
3.3.2 . . . 21
3.3.3 . . . 21

4 Towards An Emerging Science of Domains 21
4.1 Two Brief Accounts . . . 21
4.1.1 Stanisław Leśniewski’s ‘Mereology’ . . . 21
4.1.2 Bertrand Russell’s ‘Philosophy of Logical Atomism’ . . . 21
4.2 Bertrand Russell’s ‘Philosophy of Logical Atomism’ . . . 22
4.2.1 A Metaphysics and a Methodology . . . 22
4.2.2 A Logically Ideal Language . . . 22
4.2.3 A Monadistic View of Domain Modelling . . . 22
4.2.4 Discussion . . . 23
4.3 Stanisław Leśniewski’s ‘Mereology’ (III) . . . 23
4.3.1 General . . . 23
4.3.2 A Model of Mereology . . . 23
Some General Observations . . . 23
The Model . . . 24
4.3.3 Relation to The Casati Varzi Mereology [27] . . . 26
“Classical” Mereology Operators . . . 26
Discussion of Our Interpretation . . . 27
4.3.4 Discussion . . . 27
4.4 On Laws of Domain Descriptions . . . 28
4.4.1 Preliminaries . . . 28
A Spatial Phenomenon . . . 28
Suppression of Unique Identification . . . 28
Distinguishability of Simple Entities . . . 28
4.4.2 Space Phenomena Consistency . . . 29
Domain Description Law 1, Space Phenomena Consistency . . . 29

4.4.3 Unique Identifiers . . . 29
Domain Description Law 2, Unique Identifiers . . . 29
4.4.4 Unique Phenomena . . . 30
Domain Description Law 3, Unique Phenomena . . . 30
4.4.5 Space/Time Phenomena Consistency: Monotonicity . . . 30
Domain Description Law 4, Space/Time Phenomena Consistency: Monotonicity . . . 30
4.4.6 Discussion . . . 30

5 Discussion 30
5.1 Delineations of 'Philosophy', 'Theory', and 'Science' . . . 30
5.1.1 What is Philosophy ? . . . 30
Metaphysics . . . 30
Epistemology . . . 30
Logic . . . 30
5.1.2 Natural Sciences Versus Informatics . . . 31
5.1.3 What is a Theory ? . . . 31
Some Conventional Definitions of 'Theory' . . . 31
An Informatics Theory . . . 31
5.1.4 "The" Scientific Method in Natural Sciences . . . 31
5.2 What is A Philosophy of Informatics ? . . . 32
5.2.1 Ontology of Informatics . . . 32
5.2.2 Epistemology of Informatics . . . 32
5.2.3 Wider Perspectives . . . 32
5.3 What is a Theory of The Science of Informatics ? . . . 33

6 Closing 33

7 Bibliographical Notes 33

A An Example Domain Description: Railway Nets 39
Last page . . . 41

A Caveat

This paper has an example. It appears in Appendix A. An oral presentation of (as subset) of this paper will start with this example. The mathematical notation will not be explained. The formulas will not be "read". Instead we shall comment on what the example wishes to communicate: categories of real domain phenomena and concepts, signatures of predicates over these, etcetera. The presentation of this example will thus be very cursory. But it will be assumed that a reader of this paper has had a brief look at the example (thus, please turn to Pages 39–41).

1 Introduction s7

Thinkers, like Bertrand Russel, Sir Karl Popper and others, have reflected on Philosophies of Mathematics, of Sciences, etc. A new scientific field have emerged in the last 50 years: It is that of Informatics. It is claimed different from those of the natural sciences. We shall attempt a delineation of this field, and we shall attempt to identify components of a 'Philosophy of Informatics' and a 'Theory of a Science of Informatics'.

1.1 Informatics s8

To us informatics consists of the fields of (i) computer science, (ii) computing science, (iii) software engineering, (iv) information technology (IT) and (v) applications of (i–iv) to human domains.

Let me first explore, in one small subsection each, each of these five fields.

(In the following we shall assume that you have just a very superficial understanding of what a computer is and what communication is (C&C).)

1.1.1 Computer Science s9

By computer science we understand the knowledge and the study of the "things" that can "reside inside" computers and communication.

We leave formally unexplained what we mean by "things" and "reside inside"

s10 Informally "things" are *data*, *computations* and *communications*.

What is this *knowledge* and this *study* about ?

Roughly, they are about *properties:* What is data ? What do we mean by computing device ? (We many mean such things as a Turing Machine, λ–Calculus reduction rules, an Abstract State Machine et cetera.) What do we mean by computation ? (We many mean such things as a terminating process, or a never-ending process, et cetera.) What is an algorithm ? What is computable ? What do we mean by computational complexity ?

1.1.2 Computing Science s11

By computing science we understand the knowledge and the study of how to construct the "things" s12 that can "reside inside" computers and communication.

What do we mean by **"construction"** ? We mean the engineering task of understanding the **domain** in which the problem resides, prescribing the **requirements** to what is to be "constructed", and **designing** cum programming the **software** the algorithm and the data structures that shall be executed upon and stored by the computer.

• • •

Discussion: *The distinction made between (i) knowledge and study of properties of and (ii) how to construct the "things" that can exist (i.e., "reside") inside computers, that distinction can be discussed. In computer science researchers propose new models of computing devices and computation and study these and existing models. In computing science researchers propose aspects of programming methodology (principles, techniques and tools) and study these and existing programming principles, techniques and tools. Computer science can be said to be more theoretical: the proposed models of computing devices and computation exist independent of the software engineer. The proposed programming methodology principles, techniques and tools are explicitly meant to be used by software engineers. In computer science one is interested in theorems about what has been constructed. In computing science one is interested in the construction process (which may entail proving properties of intermediate and final results of that process).* •

1.1.3 Software Engineering s13

By software engineering we understand an applied form of computing science: the practical aspects of how people, in groups, effect the principles of, use the techniques for, and apply the tools for s14 describing the domain, prescribing the requirements, and designing the software

Please observe that the software engineer creates documents: domain descriptions, requirements prescriptions, software designs, often in several stages of refinement from property-oriented abstrac- s15 tions via model-oriented specifications to machine code, the latter executable by machine.

These documents are texts (and diagrams), that is, syntactic things, but having clear semantics and, oftentimes also possessing proofs. And the "lowest", most concrete level of these documents serve as the basis for computations (and communications).

1.1.4 IT: Information Technology s16

By IT we mean the **hardware** of computing and communications devices including their input/output **devices** and the base **software** upon which end-user applications are based.

Discussion: *Our delineation of what IT is can also be debated. We have chosen this definition for the following reasons: The hardware devices all satisfy laws of physics and are designed according to electro-mechanical, electrical, electronic, opto-electronic, etc., principles and techniques and* *by means of corresponding tools. Concern for IT is "measured" in terms of size ("the-smaller-the-better"), capacity ("the higher-the-better"), speed ("the-faster-the-better"), and cost ("the lower-the-better"). Concern for software is, correspondingly, "measured" in terms of "is it the right software for the user, that is, does it meet customer expectations" and "is the software right, that is, correct (no 'bugs')". It is for these reasons that we wish to make as clear a distinction between hardware ('IT') and software ('Applications').* •

1.1.5 Applications s17

By an application (or, more colloquially, an IT application) we understand a **machine**, that is: an end-user software system "residing" in a specific domain more-or-less tailored to those end-users' needs with all of this "running" on some IT.

This characterisation leaves undefined what we mean by (i) 'residing', (ii) the 'specific domain', (iii) 'tailored' (etc.) and (iv) "running". To this we turn now. s18

Discussion: *Let us restrict, just for the sake of "simplicity" (!) the applications to be more-or-less tailor-made to the specific customer. Example applications could be: (a) a hospital patient management system, or (b) an oil pipeline monitoring and control system, or (c) a railway train traffic, signal and rail track monitoring and control system.*

(i) For the application to 'reside' means that a sizeable number of application stake-holders are aware of the computerised application, that is, uses it and/or are affected by it.

(ii) The 'specific domain' is that of one of those listed above (a–c) and the stake-holders are primarily, that is, most of the time that they use IT, using this system — and not also some other computing systems such as a text-editing system (e.g., `MS Word, XLS` or other such "of-the-shelf" mass-software), some `Internet` browser, or other, to any significant extent.

(iii) The 'tailoring' finally means that the computing systems as used is thought of, by its users, as a system specifically designed for them and not for some other enterprise of the same category.

(iv) By the "running" on IT we mean that there is an application platform consisting of various forms of IT devices and that these devices provide the ultimate basis for computation and communication.

• • •

So far, in this 'discussion' we have set the stage for what this 'discussion' point is to address, namely that there are two characteristics of many applications, such as their users, mostly unwittingly, experience these applications:

*(α) **Anthropomorphisation:** Software technology, that is, applications, when being used, very often, in the interaction between the human users and the technology, the hardware and the software (i.e., the machine), lead the human users to ascribe human attributes to the machine. Even programmers say, when explaining computer code or even some abstract specification, "here the program does so-and-so, et cetera".*

*(ω) **Intellectual Versus Materiel Universe:** Informatics is a universe of intellectual quality: the right software, that is, allowing pleasing use of the application and fit, "hand-in-glove", with the domain; as well as the software being right, that is, correct with respect to requirements.* Classical technology, including now also IT, is a universe of material quantity: smaller-and-smaller physical size, larger-and-larger (storage) capacity, faster-and-faster computations and lower-and-lower costs. •

1.2 Structure of Paper s19

We have set the stage. Now let us outline the bulk of this paper. Some of the 'earlier' sections do not themselves contain material in the nature of 'philosophy' or 'theory of science', but their contents serve as a necessary, and we think, minimum background for our subsequent "musings". s20

In Sect. 2 we cover just the domain engineering phase (Sects. 2.3) of software engineering. The 'domain engineering' phase is where we, in particular, in Sect. 3–4, will discover issues of 'philosophy' and 'science of informatics.

In Sect. 3 we cover some descriptional aspects of computing science cum software engineering. This section proposes a set of meta-types, meta-predicates, meta-observers and meta-axioms that represent useful principles and techniques for describing domains. We shall, already in this section, touch upon a number of 'philosophy and theory of science of informatics' issues. s21

Section 4 is the main section of this paper — the section in which we discuss but a few issues of a 'philosophy and of theory science of informatics'. Notably these are based in Bertrand Russell's 'Philosophy of Logical Atomism' and Stanisław Leśniewski's 'Mereology'.

Section 5, in effect, closes this paper by "casting a wider net of 'philosophy and science of informatics': which qualities could characterise such a philosophy and such a theory of science.

2 Software Engineering s22

2.1 Paraphrasing 'Software Engineering'

By **software engineering**, to paraphrase Sect. 1.1.3 in the light of its discussion, we understand the actions of the three phases: **domain engineering:** describing the domain *as it is*, **requirements engineering:** prescribing the machine *as we would like it to be*, and **software design:** specifying the software *as we intend to implement it*.

2.2 Documents – Documents – Documents s23

The software engineer produces documents: domain descriptions, requirement prescriptions and software specifications.

Both informal: precise, say, English narratives and formal, mathematical/logical texts. Hence the software engineer must be versant in syntax, semantics and pragmatics. s24

Thus the software engineer expresses precise mathematical properties with no reference to laws of physics. Other engineers build on laws of physics (as expressed in mathematics), but they do not express new laws of the natural science.

The most concrete level of software engineering documents "miraculously execute" on computers ! The concrete other engineering documentsusually diagrams, drawings are then the basis for laborious production processes involving costly tools and salaried labourers before the final result, a technology, emerges. s25

Discussion: *The physical artifacts, i.e., the technologies that derive from the physical sciences, including chemistry, can be measured: tolerance or accuracy numbers can be established between the design prescriptions and each actual instantiation of the prescribed technology. And these instances are subject to statistical variance with respect to these numbers, to production errors and to wear-and-tear. Not so for software: Either software is correct with respect to requirements prescriptions or it is not correct. Software does not "age" with use. But new software versions replace older ones (software doesn't change !).*

Thus one is put in an awkward position when comparing technologies derived from laws of the natural sciences with respect to software ("technologies") derived from mathematical logical domain descriptions and requirements prescriptions. Basically one cannot compare these two kinds of technologies other than just done !

I think that these differences call for a different approach to a possible philosophy and a theory of a science of informatics. • s26

2.3 Domain Engineering s27

We shall only consider the main stages of domain acquisition and domain modelling, that is, finding out about the domain and describing "that" domain.

2.3.1 Domain Acquisition: Domain Models and Their Acceptance s28

Some physicists make experiments and, as a result of their analysis, propose mathematical models of the phenomena studied; other physicists "massage" such mathematical models and conjecture new, usually more general and abstract mathematical models.

Domain engineers[3] also "experiment": they read about the domain, they elicit information about the entities: phenomena and concepts, of the domain, when possible they "immerse" themselves in the domain, and they rough sketch formalise fragments of the domain as they find it. s29

Acceptance of the conjectures of physicists emerges as the result of the "socialisation' process of other physicists criticizing the conjectures and/or replacing them with other conjectures. and as the result of other physicists validating the conjectured models by basing own, more specialised models on these, and by showing that previously unknown properties of physical phenomena can be verified from these, or new, hitherto undiscovered (properties of physical) phenomena can be predicted. s30

"Cementation" of physical models takes place as broader acceptance and their 'entry' into monographs and textbooks.

Acceptance of domain models emerges as larger and larger domain descriptions are published and are referred to in other, published domain descriptions.

Unlike conjectured physics models — whose "behaviour", that is, whose expression of laws of physics, can be checked to be in (reasonably approximate) agreement with a perceived actual world — domain models, since they do not rely on laws of physics, but on more-or-less inexpressible laws of human behaviour, cannot be "validated" the same way as models of physics.

Discussion: *Above we have only very briefly touched upon the processes of acceptance of scientific versus domain engineering models. Basically we suggest the adage of the free interpretation of Imre Lakatos [58] and Sir Karl Popper [69, 70, 72]: there are no theories, there are no proofs, there may be bold conjectures, and there will be sad refutations.* • s31

2.3.2 Domains: Descriptions and Models s32

For pragmatics reasons we "sub-divide" the task of constructing a domain deswcription in stages: intrinsics, support technologies, management & organisation, rules & regulations, scripts: licenses, contracts, and human behaviour.

Domain Intrinsics By the intrinsics of a domain we shall understand those phenomena and concepts that appear in all other facets and without which these other facets cannot be described. s33

That is, the intrinsics of a domain are the very basic facts of the domain. s34

Some examples of domain intrinsics are:

- **Railway Systems:** the (i) railway net [but only] with its (ii) stations and (iii) tracks between stations, the (iv) trains, the (v) passengers, (xiv) strain start, (xviii) the change of a rail track signal from stop to go, and (xxi) train journey from one station to the next.
- **Health-care:** (vi) healthy and (vii) sick people, (viii) medical staff, (ix) medical clinics, (xv) singular treatment actions of sick people, (xix) death of a patient, and (xxii) a "full" hospitalisation of a patient.
- **Container Lines:** (x) containers, (xi) contain vessels, (xii) the ocean, (xiii) container terminal ports, (xvi) arrival of a container vessel at a container terminal port, (xvii) movement of containers between container vessels and container terminal ports, (xx) damage to a container, (xxiii) container "journey" from port of origin to port of destination.

Only "bare-bone" aspects of the above entities are to be considered. Example items (i-xiii) are of simple entities, (xiv-xvii) are of actions, (xviii-xx) of events, and (xxi-xxiii) of behaviours

[3]– typically for such domains as air traffic, banking, container lines, (other) financial service institutions, health care, manufacturing, oil pipeline, railways, etcetera

s35 **Domain Support Technologies** By the support technologies of a domain we shall understand the actors, whether humans or "hard" technology apparatus like mechanical, electro-mechanical, electronic and IT devices that effect, i.e., represent and/or carry out, the simple entities, actions s36 and events of the domain.

Some examples of domain support technologies are:

- **Railway Systems:** electro-mechanical or electronic + electro-mechanical (i.e., interlock) switches, software (or people) for timetable scheduling,
- **Health-care:** software (or people) for scheduling patient operations, blood pressure measurement instrument, MR scanner.
- **Container Lines:** software (or people) for (near-optimal) container stowage planning, ship to quay container crane.

s37 **Domain Management and Organisation** By domain management we mean people (i) who determine, formulate and thus set standards (cf. rules and regulations, a later lecture topic) concerning strategic, tactical and operational decisions; (ii) who ensure that these decisions are passed on to (lower) levels of management, and to floor staff; (iii) who make sure that such orders, as they were, are indeed carried out; (iv) who handle undesirable deviations in the carrying out of these orders cum decisions; and (v) who backstop complaints from lower management levels and from s38 floor staff.

By domain organisation we mean (vi) the structuring of management and non-management staff levels, (vii) the allocation of strategic, tactical and operational concerns to within management and non-management staff levels, (viii) and hence the "lines of command": who does what and s39 who reports to whom — administratively and functionally.

Some examples of domain management and organisation are:

- **Railway Systems:** station masters who, amongst other things, decide when trains have to depart, area managers who co-ordinate line traffic resources (staff rostering, train maintenance and service, etc.), engine men who must obey track signals, etc.
- **Health-care:** national health service directors who monitor and control financial resources to clinics, hospitals, etc., hospital managers who allocate and schedule medical staff (rostering, etc.), etc.
- **Container Lines:** container line directors who monitor and control financial resources, charter or commission the building of container vessels, container vessel captains who monitor and control vessel resources and interact with container terminal port management, etc.

s40 **Domain Rules and Regulations** Human stake-holders act in the domain, whether clients, workers, managers, suppliers, regulatory authorities, or other. Their actions, oftentimes aided by technology, are guided and constrained by rules and regulations.

By a **domain rule** we mean some text which prescribes how people or equipment are expected to behave when dispatching their duty, respectively when performing their functions.

By a **domain regulation** we mean some text which prescribes what remedial actions are to be s41 taken when it is decided that a rule has not been followed according to its intention.

Some examples of domain rules and regulations are:

- **Railway Systems.** *Rule* (in China): No two trains to arrive or depart[4] from any train station in any 2 minute interval. *Regulation* (in China): Dismissal.

[4]Arrival and departure are behaviours and the two minute interval must be between the time intervals of these behaviours.

- **Health-care.** *Rule* (in Denmark): hospital patient is to give their name and central personal registration number to the medical whenever that staff performs a hospitalisation related action. *Regulation*: some disciplinary action.
- **Container Lines:** *Rule*: Containers must be stowed on-board a vessel according to a number of weight, spread-of-fire and 'explosives' constraints. *Regulation*: some disciplinary action.

Domain Scripts: Licenses and Contracts By a **domain license** language we mean the definition s42
of a set of rules & regulations where these licenses when issued and actions when performed are bound by the rules & regulations and have morally obliging power.

By a **domain contract** language we mean a domain license language whose licenses and actions have legally binding power, that is, their issuance and their invocation may be contested in a court of law. s43

Some examples of domain scripts (licenses and contracts) are:

- **Railway Systems.** *License:* a timetable. *Contract:* a ticket.
- **Health-care.** *License:* educational and training degrees such as for those of medical doctors, nurses, etc. *Contract:* a health insurance.
- **Container Lines.** *License:* a voyage schedule. *Contract:* a bill-of-lading.

Domain Human Behaviour By domain human behaviour we understand any of a **quality spectrum** s44
of carrying out assigned work: from (i) **careful, diligent** and **accurate**, via (ii) **sloppy** dispatch, and (iii) **delinquent** work, to (iv) outright **criminal** pursuit. s45

Some examples of domain human behaviours are:

- **Railway Systems:** train engine man when *diligent:* obeying all signal settings all the time; when *sloppy:* missing a signal setting every other year; when *delinquent:* missing a signal setting every two weeks; and when *criminal:* deliberately disregarding a signal setting.
- **Health-care:** medical nurse, when administering medicine to a patient when *diligent:* always remembering to ask for name and CPR number; when *sloppy:* forgetting to ask one time out of at least 300 times; when *delinquent:* forgetting to ask one time out of 20 times; and when *criminal:* mostly not asking for this information.
- **Container Lines:** a container stowage planner when *diligent:* always, to the the best of their ability. trying to follow all stowage rules; when *sloppy:* occasionally, usually not deliberately, and in one out of around, say, 100 times, skipping such a rule; when *delinquent:* regularly and deliberately skipping such a rule; and when *criminal:* deliberately, and quite more often, skipping such rules.

2.3.3 Discussion s46

$\boxed{\text{To be written 1/3 page}}$

3 A Description Ontology s47

3.1 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 2.3.2 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"*. which we shall apply in the description of domain phenomena and concepts. s48

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' [4, 28, 31, 45, 46, 47, 48, 91].

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.

3.2 Categories, Predicates and Observers for Describing Domains s49

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 [9, 12, 15, 18, 19]. 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 [89, John Sowa, 1999] than the one put forward in this paper.

3.2.1 The Hypothetical Nature of Categories, Predicates and Observers s50

Categories In the following we shall postulate some categories, that is, some meta-types:

categories
$\mathbb{ALPHA}, \mathbb{BETA}, \mathbb{GAMMA}$

s51 What such a clause as the above means is that we postulate that there are such categories 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, fine. In the end, which we shall never reach, those aspects of a, or the domain science, may "survive". If not, not !

s52 **Predicates and Observers** With the categories just introduced we then go on to postulate some predicate and observer functions. For example:

predicate signatures
is_$\mathbb{ALPHA}$: "Things" $\rightarrow$ **Bool**
is_$\mathbb{BETA}$: "Things" $\rightarrow$ **Bool**
is_$\mathbb{GAMMA}$: "Things" $\rightarrow$ **Bool**
observer signatures
obs_$\mathbb{ALPHA}$: "Things" $\tilde{\rightarrow}$ $\mathbb{ALPHA}$
obs_$\mathbb{BETA}$: $\mathbb{ALPHA}$ $\tilde{\rightarrow}$ $\mathbb{BETA}$
obs_$\mathbb{GAMMA}$: $\mathbb{ALPHA}$ $\tilde{\rightarrow}$ $\mathbb{GAMMA}$

s53 So we are "fixing" a logic !
The "Things" clause is a reference to the domain under scrutiny. Some 'things' in that domain are of category $\mathbb{ALPHA}$, or $\mathbb{BETA}$, or $\mathbb{GAMMA}$. Some are not. It is then postulated that from such things of category $\mathbb{ALPHA}$ one can observe things of categories $\mathbb{BETA}$ or $\mathbb{GAMMA}$. Whether this is indeed the case, i.e., that one can observe these things is a matter of conjecture, not of proof.

s54 **Meta-Conditions** Finally we may sometimes postulate the existence of a meta-axiom:

meta condition:
Predicates over $\mathbb{ALPHA}, \mathbb{BETA}$ and $\mathbb{GAMMA}$

Again, the promulgation of such logical meta-expressions are just conjectures, not the expression of “eternal” truths.

Discussion So, all in all, we suggest four kinds of meta-notions: s55

- categories,[5]
- is_Category (and later: obs_Property), predicates,[6]
- obs_Category (and later: obs_Attribute) observers,[7] and
- meta-conditions.[8]

s56

The **category** [**type**] A, B, ..., is_A, is_B, ... obs_A, obs_B, ... **meta-condition** [**axiom**] predicate notions derive from McCarthy’s *analytic syntax* [63]. In that paper McCarthy also suggested a *synthetic syntax* constructor function: mk_A, ...At present, we find no need to introduce this *synthetic syntax* constructor function. A basic reason for this is that we are not constructing domain phenomena. The reason McCarthy (and computing science) needed the *synthetic syntax* constructor functions is that software is constructed.

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.* •

3.2.2 Entities s57

What we shall describe is what we shall refer to as entities. In other words, there is a category and meta-logical predicate $\mathbb{ENTITY}$, is_$\mathbb{ENTITY}$. The is_$\mathbb{ENTITY}$ 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_$\mathbb{ENTITY}$: “Thing” $\rightarrow$ **Bool**
meta condition:
  $\forall$ e:$\mathbb{ENTITY}$ • is_$\mathbb{ENTITY}$(e)

Discussion: *When we say “things”, or entities, others may say ‘individuals’, ‘objects’, or use other terms.*

The meta-predicate is_$\mathbb{ENTITY}$ 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. In Sect. 4.2 we shall discuss is_$\mathbb{ENTITY}$. • s58

[5]See the example of Appendix A. Here the categories are now called types — since it is a specific domain descriptions, and are specified by a clause of the form **type** A, B, C.

[6]See the example of Appendix A. Here there is no is_A etc. clauses. Instead RSL uses the following clauses **let**

• • •

By introducing the predicate is_$\mathbb{ENTITY}$ we have put the finger on what this section, that is, Sect. 3, 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.

3.2.3 Entity Categories s59

We postulate four entity categories:

category:
  $\mathbb{SIMPLE\_ENTITY}$, $\mathbb{ACTION}$, $\mathbb{EVENT}$, $\mathbb{BEHAVIOUR}$

Simple entities are **phenomena** or **concepts**. **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. A subset of simple s60 domain entities form a **state**. **Actions** are the result of applying functions to simple domain entities and changing the **state**. **Events** are **state** changes that satisfy a predicate on the 'before' and 'after s61 states'. **Behaviours** are sets of sequences (of sets of) actions and events.

category:
  $\mathbb{ENTITY} = \mathbb{SIMPLE\_ENTITY} \cup \mathbb{ACTION} \cup \mathbb{EVENT} \cup \mathbb{BEHAVIOUR}$

Discussion: *The four categories of entities may overlap.* •
With each of the four categories there is a predicate:

predicate signature:
  is_$\mathbb{SIMPLE_ENTITY}$ "Thing" $\rightarrow$ **Bool**
  is_$\mathbb{ACTION}$ "Thing" $\rightarrow$ **Bool**
  is_$\mathbb{EVENT}$ "Thing" $\rightarrow$ **Bool**
  is_$\mathbb{BEHAVIOUR}$ "Thing" $\rightarrow$ **Bool**

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

  is_$\mathbb{ENTITY}$(t)

The $\cup$ "union" is inclusive:

meta condition:
  $\forall$ t:"Thing"•is_$\mathbb{ENTITY}$(t) $\Rightarrow$
    is_$\mathbb{SIMPLE_ENTITY}$(t) $\vee$ is_$\mathbb{ACTION}$(t) $\vee$ is_$\mathbb{EVENT}$(t) $\vee$ is_$\mathbb{BEHAVIOUR}$(t)

3.2.4 Simple Entities s62

Anticipating the discussion of Sects. 4.2 and 4.3 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 s63 entities. That is:

a:A, b:B, ... **in** ... **end**.
[7]See the example of Appendix A. There are several observer functions defined in that appendix.
[8]See the example of Appendix A. Here the meta-conditions are "labelled" **axioms**.

category:
$\mathbb{SIMPLE\_ENTITY} = \mathbb{ATOMIC} \cup \mathbb{COMPOSITE}$
observer signature:
is_$\mathbb{ATOMIC}$: $\mathbb{SIMPLE\_ENTITY} \to$ **Bool**
is_$\mathbb{COMPOSITE}$: $\mathbb{SIMPLE\_ENTITY} \to$ **Bool**
meta condition:
$\mathbb{ATOMIC} \cap \mathbb{COMPOSITE} = \{\}$
$\forall$ s: "Things":$\mathbb{SIMPLE\_ENTITY}$ •
is_$\mathbb{ATOMIC}$(s) $\equiv$ ∼is_$\mathbb{COMPOSITE}$(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' [85]. See Sect. 4.1.2 and Sect. 4.2.* •

Discrete and Continuous Entities We postulate two forms of $\mathbb{SIMPLE\_ENTITIES}$: $\mathbb{DISCRETE}$, such as a railroad net, a bank, a pipeline pump, and a securities instrument, and $\mathbb{CONTINUOUS}$, such as oil and gas, coal and iron ore, and beer and wine. s64

category:
$\mathbb{SIMPLE\_ENTITY} = \mathbb{DISCRETE\_SIMPLE\_ENTITY} \cup \mathbb{CONTINUOUS\_SIMPLE\_ENTITY}$
predicate signatures:
is_$\mathbb{DISCRETE_SIMPLE_ENTITY}$: $\mathbb{SIMPLE\_ENTITY} \to$ **Bool**
is_$\mathbb{CONTINUOUS_SIMPLE_ENTITY}$: $\mathbb{SIMPLE\_ENTITY} \to$ **Bool**
meta condition:
[is it desirable to impose the following]
$\forall$ s:$\mathbb{SIMPLE\_ENTITY}$ •
is_$\mathbb{DISCRETE_SIMPLE_ENTITY}$(s) $\equiv$ ∼$\mathbb{CONTINUOUS\_SIMPLE\_ENTITY}$(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.* •

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. s65

category:
$\mathbb{ATTRIBUTE}, \mathbb{NAME}, \mathbb{TYPE}, \mathbb{VALUE}$
observer signature:
obs_$\mathbb{ATTRIBUTE}$s: $\mathbb{SIMPLE\_ENTITY} \to \mathbb{ATTRIBUTE}$**-set**
obs_$\mathbb{NAME}$: $\mathbb{ATTRIBUTE} \to \mathbb{NAME}$
obs_$\mathbb{TYPE}$: $\mathbb{ATTRIBUTE} \times \mathbb{NAME} \to \mathbb{TYPE}$
obs_$\mathbb{VALUE}$: $\mathbb{ATTRIBUTE} \times \mathbb{NAME} \to \mathbb{VALUE}$
meta condition:
$\forall$ s:$\mathbb{SIMPLE\_ENTITY}$ •
$\forall$ a,a′:$\mathbb{ATTRIBUTE}$ • {a,a′}⊆obs_$\mathbb{ATTRIBUTE}$s(s)
$\wedge$ a≠a′ ⇒ obs_$\mathbb{NAME}$(a)≠obs_$\mathbb{NAME}$(a′)

s66
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. s67

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.

s68 **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'.* •

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.

s69

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 → SIMPLE_ENTITY-**set**

s70

Mereology (I) 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**. Sect. 4.3 deals with this subject with quite a different approach than the one taken here. 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**

s71

- **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. s72

  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)
  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 also just postulate the **elements of** and the **cardinality of** meta-functions. s73

- **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 ~→ list of SIMPLE_ENTITY
      pre is_LIST(s): is_COMPOSITE(s)
   obs_LENGTH: COMPOSITE ~→ Nat
      pre is_LIST(s): is_COMPOSITE(s)
meta condition:
   ∀ s:SIMPLE_ENTITY•
      is_COMPOSITE(s)∧is_LIST(s) ⇒
         obs_SIMPLE_ENTITIES(s) = elements of obs_LIST(s)
```

We also just postulate the **list of** and the **elements of** meta-functions. s74

- **Map Mereology:** The individual sub-entities of a map are "indexed" by unique definition set elements. Thus we can speak of pairings of unique map definition set element identifications and their not necessarily distinct range set elements. By a map we shall therefore understand a function, with a finite definition set, from distinct definition set elements to not necessarily distinct range elements, such that the pairs of (definition set,range) elements, which are all simple entities, can be characterised by a predicate. s75

It is this, the finiteness of maps and the (potential, but often un-expressed) predicate, which 'distinguishes' maps from functions.

Let us refer to the map as being of **category** MAP. Let us refer to the definition set elements of a map as being the DEFINITION_SET of the MAP. Let us refer to the range elements of such a map as being the RANGE of the MAP. No two definition set elements of a map, to repeat, are the same. Given a definition set element, s, of a map, m, one can obtain its IMAGE of the RANGE of m. s76

```
predicate signature:
   is_MAP: COMPOSITE → Bool
observer signatures:
   obs_MAP: COMPOSITE ~→ MAP
      pre obs_MAP(c): is_MAP(c)
   obs_DEFINITION_SET: MAP → SIMPLE_ENTITY-set
      pre obs_MAP(c): is_MAP(c)
   obs_RANGE: MAP → SIMPLE_ENTITY-set
      pre obs_MAP(c): is_MAP(c)
   obs_IMAGE: MAP × SIMPLE_ENTITY ~→ SIMPLE_ENTITY
      pre obs_IMAGE(m,d): is_MAP(m) ∧ d ∈ obs_DEFINITION_SET(m)
meta condition:
   ∀ m:SIMPLE_ENTITY•
      is_COMPOSITE(m)∧is_MAP(m) ⇒
         obs_SIMPLE_ENTITIES(m) =
            {(d,obs_IMAGE(c,d))|d:SIMPLE_ENTITY•d ∈ obs_DEFINITION_SET(m)}
```

s77

Given that we can postulate "an existence" of the obs_DEFINITION_SET and the obs_RANGE observer functions we can likewise postulate a category

```
category
    MAP = map of SIMPLE_ENTITY  into ENTITY
observer signatures
    obs_DEF_SET: MAP → set of SIMPLE_ENTITY
    obs_RNG: MAP → set of ENTITY
meta condition
    ∀ m:MAP •
        obs_DEF_SET(m) = obs_DEFINITION_SET(m)
      ∧ obs_RNG(m) = obs_RANGE(m)
```

Here, again, the **map of ... into ...** is further unexplained.

s78 We shall not pursue the notions of DEF_SET and RNG further.

- **Graph Mereology:** The individual sub-entities of a composite entity are "ordered" like elements of a graph, i.e., a net, of elements. Trees and lattices 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. s79

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

s80 In Sect. 4.3 we shall pursue quite another line of inquiry as to what establishes a mereology.

3.2.5 Actions s81

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. 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**

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 of the CARTESIAN ARGUMENT are SIMPLE STATE ENTITIES. s82 s83

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

s84
Discussion: *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. •

3.2.6 Events s85

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. s86

Examples of 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. s87

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: the domain engineer cum researcher can decide on these abstractions, these approximations. s88

category:

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

3.2.7 Behaviours s89

By a BEHAVIOUR we shall understand a set of sequences of ACTIONs and EVENTs such 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. s90

We may think of the mutually exclusive synchronisation and communication EVENTs as being designated simply by their PREDICATEs such as, for example, in CSP [52, 82, 86, 53]:

```
type A, B, C, D, M
channel ch M
value
    f: A → out ch  C
    g: B → in ch  D
    f(a) ≡ ... point ℓ_f:ch!e ...
    g(b) ≡ ... point ℓ_g:ch? ...
```

Here the zero capacity buffer communication channel, ch, express mutual exclusivity, and the output/input clauses: ch!e and ch? express synchronisation and communication. s91

The predicate is here, in the CSP schema, "buried" in the simultaneous occurrence behaviour f having "reached point" **point** ℓ_f and behaviour g having "reached point" **point** ℓ_g. s92

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. s93

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. s94

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.[9] It will bring us into a rather long discourse if we s95

[9] If it is important that ACTIONs take place over time, that is, are not instantaneous, then we can just consider

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 Benthems seminal work on the `The Logic of Time` [94]. In addition, more generally, we refer to A.N. Prior's [74, 75, 76, 77, 73] and McTaggart's works [64, 36, 81]. The paper by Wayne D. Blizard [24] proposes an axiom system for time-space.

3.2.8 Mereology (II) s96

Compositionality of Entities 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 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 s97 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 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 of several employes, etcetera. s98

The problems of compositionality of entities, whether simple, actions, events or behaviours, is was studied, initially, in [19, Bjørner and Eir, 2008]

Impossibility of Definite Mereological Analysis of Seemingly Composite Entities It would be s99 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. Section 4.3 will approach the question of mereologies of simple entities from another viewpoint than taken in the present section.

3.3 What Exists and What Can Be Described ? s100

In the previous section we have suggested a number of *categories*[10] of entities, a number of *predicate*[11] and *observer*[12] 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.

3.3.1 Description Versus Specification Languages s101

Footnotes 10–12 (Page 19) 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

ACTIONs as very simple SEQUENTIAL_BEHAVIOURs not involving EVENTs.

[10]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.

[11]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.

[12]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.

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 10–12) identifiers mean.

s102 **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.

s103 **Formal Domain Specification Languages** There are, today, 2009, 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. s104

Examples of textual, formal specification languages are

- `Alloy [57]`: model-oriented,
- `B, Event-B [1, 25, 2]`: model-oriented,
- `CafeOBJ [40, 35]`: property-oriented (algebraic),
- `CASL [5, 32, 66]`: property-oriented (algebraic),
- `DC (Duration Calculus) [101, 102]`: temporal logic,
- `RAISE, RSL [42, 7, 8, 9, 41]`: property and model-oriented,
- `TLA+ [59, 65]`: temporal logic and sets,
- `VDM, VDM-SL [21, 22, 39, 38]`: model-oriented and
- `Z [99, 51]`: model-oriented.

`DC` and `TLA+` are often used in connection with either a model-oriented specification languages or just plain old discrete mathematics notation ! s105

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

- `Petri Nets [79, 78, 80]`,
- `Message Sequence Charts (MSC) [54, 55, 56]`,
- `Live Sequence Charts (LSC) [33, 50]` and
- `Statecharts [49]`.

s106 **Discussion: "Take-it-or-leave-it !"** 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 !

3.3.2 s107

3.3.3 s108

4 Towards An Emerging Science of Domains s109

We focus on two strands of discourse: Bertrand Russell's Philosophy of Logical Atomism and Stanisław Leśniewski's 'mereology'.[13]

4.1 Two Brief Accounts

Before going into details let us review the two strands of discourse.

4.1.1 Stanisław Leśniewski's 'Mereology' s110

From ancient time and, perhaps, culminating in the 20th Century, thinkers have speculated on the logical structure of the world. Aristotle [3, Metaphysics, Book IV, Chapter 2] was perhaps the first person to consider the part-whole relationship. But it was not until the 1920s that the Polish mathematician Stanisław Leśniewski gave a formal treatment [61]. Woodger [100] and Tarski [92] made use of a specific adaptation of Leśniewski's work as a basis for a formal theory of physical things and their parts. The term 'calculus of individuals' was introduced by Leonard and s111 Goodman [60] in their presentation of a system very similar to Tarski's adaptation of Leśniewski's mereology. Slightly earlier than Leśniewski's development of his mereology, Whitehead [98] and [97, The Concept of Nature] was developing a theory of extensive abstraction. This system, according to Russell [83], was to have been the fourth volume of their Principia Mathematica, the never-published volume on geometry. Both Leśniewski [61] and Tarski [93, Foundations of the geometry of solids] have recognized the similarities between Whitehead's early work and Leśniewski's Mereology. Nelson Goodman mediated the Calculus of Individuals in [43, The Structure of Ap- s112 pearance] . Goodman had earlier used the Calculus of Individuals in his Ph.D. dissertation thesis [44, A Study of Qualities, 1940]. As in his joint paper with Leonard, he used the calculus as an addition to set theory to solve a problem known as the 'difficulty of imperfect community' (Rudolf Carnaps [26, Der Logische Aufbau der Welt]).

The field of mereology is still active, now part, mostly of computer science. Active researchers are Barry Smith [88, Mereotopology: A Theory of Parts and Boundaries], Achille C. Varzi [96, Spatial Reasoning in a Holey World[14]] and [95, On the Boundary between Mereology and Topology], and others.

4.1.2 Bertrand Russell's 'Philosophy of Logical Atomism' s113

Along a different line Bertrand Russell [85, Philosophy of Logical Atomism] (and [87, Vol. 8, Part III, Chap. 17, pp 157–244]) expressed a *metaphysics* which we summarise as:

> *the world consists of a plurality of independently existing things exhibiting qualities and standing in relations; all truths are ultimately dependent upon a layer of facts and these facts consist of either a simple particular exhibiting a quality, or multiple simple particulars standing in a relation.*

s114

Russell also expressed a *methodology* for doing philosophy:[15].

The methodology consists of a two phase process.

> *The first phase is dubbed the "analytic" phase (although it should be noted that sometimes Russell used the phrase "analysis" for the whole procedure). One begins with a certain*

[13]We spell mereology with lower case first letter as mereology is a concept that is not unique to its researchers whereas Bertrand Russell's 'Philosophy of Logical Atomism' is.

[14]holey: something full of holes

[15]http://plato.stanford.edu/entries/logical-atomism/

> *theory, doctrine or collection of beliefs which is taken to be more or less correct, but is taken to be in certain regards vague, imprecise, disunified, overly complex or in some other way confused or puzzling. The aim in the first phase is to work backwards from these beliefs, taken as a kind of "data", to a certain minimal stock of undefined concepts and general principles which might be thought to underlie the original body of knowledge.* s115
>
> *The second phase, which Russell described as the "constructive" or "synthetic" phase, consists in rebuilding or reconstructing the original body of knowledge in terms of the results of the first phase. More specifically, in the synthetic phase, one defines those elements of the original conceptual framework and vocabulary of the discipline in terms of the "minimum vocabulary" identified in the first phrase, and derives or deduces the main tenets of the original theory from the basic principles or general truths one arrives at after analysis.*

s116

Together the two, the metaphysics and the methodology expresses an essence of Russell's 'Philosophy of Logical Atomism'. We shall relate this philosophy to domain science.

4.2 Bertrand Russell's 'Philosophy of Logical Atomism' s117

4.2.1 A Metaphysics and a Methodology

Russell described his philosophy, which he referred to as 'Logical Atomism', as based on both a *metaphysical* view and a way (a *methodology*) of doing philosophy.

We refer to the indented 'metaphysics' and the 'methodology' paragraphs of Sect. 4.1.2 (Pages 21–22) for their characterisation. s118

The reason we are interested in Russell's 'Logical Atomism' is that its *metaphysical* view and its *methodology* — although meant for philosophical inquiry — very much resembles our view of the metaphysics of domains and the method by which domain analysts analyse.

We refer to [87, Vol. 8, Part III, Chap. 17, pp 157–244, same as [85]]. Russell's writings on 'Logical Atomism' contains several sub-topics. We shall next examine some of these.

4.2.2 A Logically Ideal Language s119

The analysis part of the methodology, Russell claimed, would eventually result in a minimal language containing only words for simple particulars and concepts, their simple properties, relations amongst these and logical constants — a language which could adequately capture all truths; that is, other particulars and concepts can be defined and the most general and basic principles can be derived. s120 In the minimal language the simplest of complete sentences, containing just a single predicate or verb representing a quality (i.e., an attribute) or a relation over simple entities, would be what Russell called *atomic propositions*. The truth of an atomic proposition then depends on a single atomic fact. *Molecular propositions* are then formed by combining atomic propositions using the logical connectives. *Existential* or *general propositions* are formed by replacing proper constituents of simpler propositions by variables and prefixing a universal or an existential quantifiers. s121 Hence atomic facts are at the core of Russell's metaphysics.

Instead of first using only informal language to narratively describe domain facts we advocate, motivated by Russell's Logical Atomism, both using precise formal language and to couple statements in that formal language to informal narratives — each in their way describing the domain. In our case we use a partly algebraic, that is a logical language over sorts, partly a model-oriented language over mathematical structures such as sets, Cartesians, lists, maps (i.e., definite definition set functions) and functions.

4.2.3 A Monadistic View of Domain Modelling s122

Russell opposed the "doctrine of internal relations" (every relation is grounded in the natures of the related terms) and instead expressed [84, Page 221] *"it is a common opinion … that all propositions ultimately consist of subject and predicate"*.

An interpretation of this is: *"Given, say, the proposition aRb, where R is some relation, this monadistic view will analyse this into two propositions, which we may call a_{r_1} and b_{r_2}, which give to a and b respectively, adjectives supposed to be together equivalent to R."* s123

This monadistic view is analysed in [37, J.G. Nilsson: On Reducing Relationships to Property Ascriptions] where it is *"argued that relationships may preferably be represented formally as property ascriptions."*

In Sect. 3.2.4 the monadistic view was adopted: With atomic and composite simple entities, a and b, we associated attributes, including the mereological ones of composite attributes. Let two such simple entities, a and b, be connected by a "possessing" an identifier, i_b of b, and b "possessing" an identifier, i_a of a, as in the example of Sect. A. The subjects are a and b, and the constants of a_{r_1} and b_{r_2} are i_b, respectively i_a. Thus the mereological view supports the monadistic view when expressing properties of composite entities. (Russell uses the term 'complex' where we use the term 'composite'.)

4.2.4 Discussion s124

There are other facets to Russell's work on Logical Atomism. It seems, however, to this author, that the issues we have covered in this section and which have otherwise been covered in previous and the next section suffice for now.

Our preliminary conclusions are twofold: (i) whereas mereology has very clear and focused "uses" in domain science, (ii) the "uses" of Logical Atomism is of a more meta-conceptual nature, that is, as brought out in Sect. 4.1.2's two *slant font paragraphs* on metaphysics and methodology (Pages 21–22).

4.3 Stanisław Leśniewski's 'Mereology' (III) s125

4.3.1 General

"Mereology (from the Greek μρεο, 'part') is the theory of part-hood relations: of the relations of part to whole and the relations of part to part within a whole. It is not until Leśniewski's *Foundations of a General Theory of Manifolds* (1916, in Polish) that a pure theory of part-relations was given an exact formulation. Because Leśniewski's work [62, 90] was largely inaccessible to non-readers of Polish, it is only with the publication of Leonard and Goodman's The Calculus of Individuals ([60, Goodman 1940]) that mereology has become a chapter of central interest for modern ontologists and meta-physicians."[16] s126

Our interest in mereology was first "awoken" by presentations, in the IFIP Working Group 2.3 in the 1980s and 1990s, by the late Douglas Taylor Ross[17].

We shall now present an example which is claimed to capture an essence of the kind of mereologies that are covered in [27, Casati & Varzi 1999].

4.3.2 A Model of Mereology s127

Some General Observations The example is claimed to be generic. When shown as diagrams, the boxes–within–boxes and the "fat", black connectors that "criss-cross" boxes, Fig. 1, can be shown to "mimic" the structure of such infrastructure components as: air traffic, a financial service industry, a pipeline system, a railway system, etcetera. s128

Similarly the formula parts (for boxes and connectors) thus relate to phenomena of such systems. For air traffic some boxes are aircraft, others are ground or terminal control towers, yet others are regional and continental control centers, etc. Connectors of air traffic are the radiotelephone paths that allow communication between air traffic equipment and staff. For a railway system some boxes are train stations, others are railway tracks (lines) between stations — with stations and lines consisting of embedded boxes in the form of rail units: linear, switches, crossovers, etc. Connectors compose boxes into meaningful track layouts and signalling paths.

[16]Stanford Encyclopedia of Philosophy / Wikipedia
[17]http://en.wikipedia.org/wiki/Douglas_T._Ross

s129 **The Model** We speak of systems, s:S, as *assemblies* (below referred to as a:A). From an assembly we can immediately observe, obs_Ps, a set of particulars. Particulars are either assemblies or *units*. For the time being we do not further define what units are.

type
  S = A, U, P = A | U
value
  obs_Ps: A → P-**set**

Particulars observed from a assembly are said to be immediately *embedded* (or *within*) in that assembly; and two or more particulars observed from an assembly are said to be immediately
s130 *adjacent* to one another.

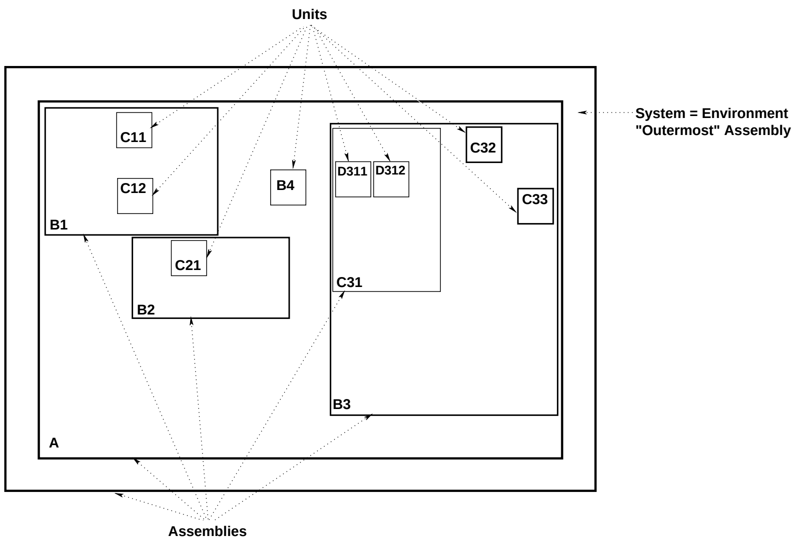

Figure 1: Assemblies and units embedded in an *environment*

s131
Figure 1 illustrates a hypothetical composite entity[18].
  Embeddedness and adjacency generalises to transitive relations.
  All particulars observable from a system are distinct.
  Given obs_Ps we can define a function, xtr_Ps, which applies to an assembly, a, and which extracts all particulars properly embedded in a. The functions obs_Ps and xtr_Ps define the meaning of *proper embeddedness*.

value
  xtr_Ps: A → P-**set**
  xtr_Ps(a) ≡
    let ps = obs_Ps(a) **in** ps ∪ ∪{xtr_Ps(a′)|a′:A•a′ ∈ ps} **end**

s132
Particulars have *unique identifiers*, PI.

type
  PI

[18]Figures 1 and 2 on the next page only serve to illustrate the formulas. In a domain description we do not give instances of figures of simple entities such as the generic ones here. Instead we present sorts (as here, where L, H, LI, HI, PI, K and KI are sorts, and types (as here, where P is defined in terms of L and H, and where axioms using observer functions impose an appropriate syntax on how simple entities are composed.

```
value
   obs_PI: P → PI
axiom
   ∀ a:A •
      let ps = obs_Ps(a) in
      ∀ p′,p″:P • {p′,p″}⊆ps ∧ p′≠p″ ⇒ obs_PI(p′)≠obs_PI(p″) ∧
      ∀ a′,a″:A • {a′,a″}⊆ps ∧ a′≠a″ ⇒ xtr_Ps(a′)∩ xtr_Ps(a″)={} end
```

s133
We shall now add to this a rather general notion of particulars being otherwise related. That notion is one of *connectors*, k:K.

Connectors may, and usually do provide for connections — between particulars. A connector is an ability be be connected. A connection is the actual fulfillment of that ability. Connections are relations between two particulars. Connections "cut across" the "classical" *particulars being part of the (or a) whole* and *particulars being related by embeddedness or adjacency*. s134

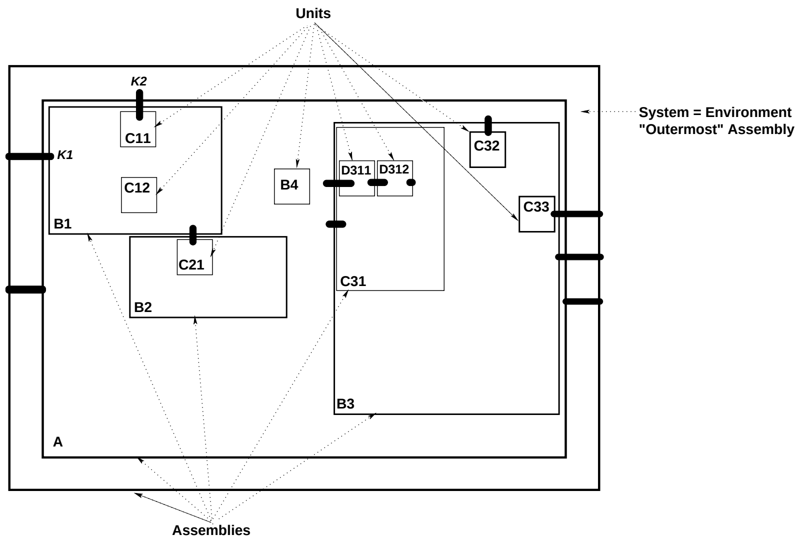

Figure 2: Assembly and unit connectors

s135
Figure 2 "repeats" Fig. 1 but "adds" connectors. The idea is that connectors allow a assembly to be connected to any embedded particular, and allow any two (transitively) adjacent particulars to be connected.

In Fig. 2 assembly A is connected, by *K2*, (without, as we shall later see, interfering with assembly B1), to part C11; the "outermost" assembly is connected, by K1 to B1, etcetera. s136

From a system, i.e., an assembly, we can observe, obs_Ks, all its connectors. From a connector we can observe, obs_KI, its unique connector identifier, KI, and the set of two (like an unordered pair of) identifiers, obs_PIp, of the particulars that the connector connects, All particular identifiers of system connectors identify particulars of the system. All observable connector identifiers of particulars identify connectors of the system. s137

```
type
   K
value
   obs_Ks: S → K-set
   obs_KI: K → KI
   obs_PIp: K → PI-set
   obs_KIs: P → KI-set
```

axiom
  ∀ s:S,k:K • k ∈ obs_Ks(s) ⇒ ∃ p:P • p ∈ xtr_Ps(s) ⇒ obs_PI(p) ∈ obs_PIp(k),
  ∀ s:S,p:P • ∀ ki:KI • ki ∈ obs_KIs(p) ⇒ ∃! k:K • k ∈ obs_Ks(s) ∧ ki=obs_KI(k)

This model allows for a rather "free-wheeling" notion of connectors one that allows internal connectors to "cut across" (even transitively) embedded and (transitively) adjacent particulars. s138

We may need to define an auxiliary function. xtr∀KIs(p) applies to a system and yields all its connector identifiers.

value
  xtr∀KIs: S → KI-**set**
  xtr∀Ks(s) ≡ {obs_KI(k)|k:K•k ∈ obs_Ks(s)}

s139

There is an aspect of assemblies and units that we have not mentioned. You may, "mereologically speaking", think of that aspect as the "white space" within the borders of a unit box, and within the borders of an assembly box exclusive of that assembly box's embedded assemblies and units. Let us associate a notion of state, i.e., a state, that is, set of static and dynamic attributes of assemblies and units, with that "white space" ! s140

type
  Σ
value
  obs_Σ: P → Σ

In axiomatic mereology there is a notion of overlap. Overlap, to us, has to do with two "white spaces" of distinct particulars. Connectors between such particulars can then be interpreted as also providing for overlaps.

The above model is now claimed to be a model of axiom systems proposed in the literature [27, Casati & Varzi 1999].

• • •

s141

This ends our model of a concept of mereology. The particulars are those of assemblies and units. The relations between particulars and the whole are, on one hand, those of embeddedness and adjacency, and on the other hand, those expressed by connectors: relations between arbitrary `part`iculars.

4.3.3 Relation to The Casati Varzi Mereology [27] s142

"Classical" Mereology Operators Typically `part/whole` relations for mereologies in [27, Casati & Varzi 1999] are:

- $\mathcal{P}xy$: x is a `part` of y,
- $\mathcal{PP}xy$: x is a proper `part` of y,
- $\mathcal{O}xy$: x and y overlaps and
- $\mathcal{U}xy$: x and y underlap.

We refer to [27, Casati & Varzi 1999] for axioms over these operators. We now give but one of many possible interpretations of the above `part/whole` relations. The interpretation is with respect to the model given in Sect. 4.3.2. We apologize for the "double" use of the term 'part': s143 when we use the *slanted sans serif font 'part'* we refer to a p:P, and when we use the `teletype font 'part'` we me the relationship designated by $\mathcal{P}$ or $\mathcal{PP}$. The mereology operator $\mathcal{P}$ is taken as a primitive. The other mereology operators, $\mathcal{PP}$, $\mathcal{O}$ and $\mathcal{U}$ are defined in terms of $\mathcal{P}$. s144

- $\mathcal{P}xy$: x is a `part` of y.[19]

[19]The • (bulleted) statement(s) expresses an informal definition of the mereology operator.

$\star$ A *part* is a part of itself; attributes of a *part* are parts of that *part*.[20]

- $\mathcal{PP}xy$: x is a proper part of y: $\mathcal{PP}xy \leftrightarrow (\mathcal{P}xy \wedge \sim \mathcal{P}xy)$.

  $\star$ *Part*s of an assembly are proper parts of that assembly. Any proper subset of attributes of a *part* are proper parts of that *part*.

s145

- $\mathcal{O}xy$: x and y overlaps: $\mathcal{O}xy \leftrightarrow \exists z[\mathcal{P}zx \wedge \mathcal{P}zy]$.

  $\star$ Two *part*s that are connected overlap. If two or more attributes of a *part* are not (mutually) independent, that is, these dependent attribute values stand in some relation to one another, then these attributes overlap.

- $\mathcal{U}xy$: x and y underlap if there exists an object z such that x and y are both parts of z: $\mathcal{U}xy \leftrightarrow \exists z[\mathcal{P}xz \wedge \mathcal{P}yz]$

  $\star$ If two or more *part*s are contained in some assembly, a, then these contained *part*s underlap with a.

Discussion of Our Interpretation First we remind the reader that we have given but one of many possible interpretations of the part/whole relations. s146

We could have given other interpretations; for example: (i) we could have omitted any reference to *part* attributes; (ii) we could have restricted overlap to relate to only specifically characterised *connectors*; (iii) or both (i–ii).

The scope of our discussion will now be enlarged to cover other kinds of mereologies.

4.3.4 Discussion s147

We have partially covered the mereological system of [27, Casati & Varzi, 1999]. There are other mereological systems.

In [29, Bowman L. Clarke, 1981] (*A Calculus of Individuals Based on 'Connection'*) a mereology operator, $\mathcal{C}$, for 'connected'[21], is the primitive in terms of which the above operators ($\mathcal{P}, \mathcal{PP}, \mathcal{O}$, and $\mathcal{U}$) can be defined. The intuition behind the $\mathcal{C}$onnected operator, however, makes it of less interest to us: that intuition implies a notion of points which we do not wish to bring into our models of man-made infrastructure components. Bowman L. Clarke's system allows definition of external and internal, as well as tangential connectedness. [30, Bowman L. Clarke, 1985] (*Individuals and Points*) further elaborates on this, for our purpose, "uninteresting" concept of points. s148

In [96, Achille C. Varzi, 1993] (*Spatial Reasoning in a Holey*[22] *World*), in [95, Achille C. Varzi, 1994], (*On the Boundary between Mereology and Topology*) and in [88, Barry Smith, 1996] (*Mereo-topology: A Theory of Parts and Boundaries*), extensions to the operators covered above (i.e., ($\mathcal{P}, \mathcal{PP}, \mathcal{O}$, and $\mathcal{U}$) are applied to cover topological spaces. It would be interesting to study these papers further in order to determine their relevance to the modelling of the man-made infrastructure components in which we are interested. s149

In closing this section on 'mereology' we claim (i) that our model of mereology is in line with main-stream axiom systems for the kind of mereologies that Stanisław Leśniewski studied, (ii) that that model covers of a large class infrastructure components which we have so far modelled[23], and (iii) that, apparently, we do not need a more sophisticated concept of mereology than covered by [27].

[20] The $\star$ (starred) statement(s) interprets the $\bullet$ (bulleted) statement(s) in terms of the model of mereology given in Sect. 4.3.2.

[21] The $\mathcal{C}$onnected operator is not to be confused with the connectors, k:K, of our model.

[22] holey: something full of holes

[23] [6, The Market], [23, Railways], [11, Financial Service Industry], [10, A Container Line Industry], [17, Logistics], [13, Oil Pipelines], etcetera.

4.4 On Laws of Domain Descriptions s150

4.4.1 Preliminaries

A Spatial Phenomenon Some phenomena (p:P) enjoy the meta-linguistic "$\mathcal{L}$ property", $\mathcal{L}$ for $\mathcal{L}$ocation. Let any phenomenon and its derived concepts be subject to the meta-linguistic predicate, has_$\mathcal{L}$, and function, obs_$\mathcal{L}$: s151

```
type
    P, C, $\mathcal{L}$, E = P|C
value
    $\emptyset$:$\mathcal{L}$
    has_$\mathcal{L}$: E $\rightarrow$ Bool
    obs_$\mathcal{L}$: E $\overset{\sim}{\rightarrow}$ $\mathcal{L}$, pre obs_$\mathcal{L}$(e): has_$\mathcal{L}$(e)
    $=,\neq$: $\mathcal{L} \times \mathcal{L} \rightarrow$ Bool
    $\sqcap,\sqcup$: $\mathcal{L} \times \mathcal{L} \rightarrow \mathcal{L}$
    $\sqsubset,\sqsubseteq$: $\mathcal{L} \times \mathcal{L} \rightarrow$ Bool
axiom
    $\forall$ e,e′:E•has_$\mathcal{L}$(e)$\wedge$has_$\mathcal{L}$(e′)$\wedge$e$\neq$e′ $\Rightarrow$ obs_$\mathcal{L}$(e)$\sqcap$obs_$\mathcal{L}$(e′)=$\emptyset$
```

s152

The location point space, $\mathcal{L}$, the empty location, $\emptyset$, and the operations $=, \neq, \sqcup, \sqcap, \sqsubset$ and $\sqsubseteq$ are defined using standard mathematical topology.

The axiom expresses that two observable, i.e., phenomenological (simple) entities that both enjoy the has_$\mathcal{L}$ property, have distinct, non-overlapping locations. s153

We consider $\mathcal{L}$ to be an attribute of those phenomena (and concepts) which satisfy the has_$\mathcal{L}$ property.[24]

Universals, that is, attributes of entities (that may enjoy the has_$\mathcal{L}$ property), do not satisfy the has_$\mathcal{L}$ property.

Examples of observable phenomena, that do not enjoy the has_$\mathcal{L}$ property, are: voltage, a person's age, the colour red, etc. □

s154 **Suppression of Unique Identification** When comparing, for example, two simple entities one is comparing not only their attributes but also, when the entities are composite, their sub-entities. Concerning unique identifiers of simple entities we have this to say: We can decide to either include unique identifiers as an entity attribute, or we can decide that such identifiers form a third kind of observable property of a simple entity the two others being ("other") attributes — as we see fit to define and the possible sub-entities of composite entities. s155

Either way, we need to introduce a meta-linguistic operator[25], say

$$\mathcal{S}_I\text{: Simple\_observable\_entity\_value} \rightarrow \text{Anonymous\_simple\_entity\_value}$$

The concept of an anonymous value is also meta-linguistic. The anonymous value is basically "the same, i.e., "identical" value as is the simple entity value (from which, through $\mathcal{S}_I$, it derives)[26] with the single exception that the simple entity value "possesses" the unique identifier of the observable entity value and the anonymous entity value does not.

s156 **Distinguishability of Simple Entities** When we wish to distinguish one simple entity phenomenon from another then we say that the two ("the one and the other") are distinct. To be distinct to us means that the two phenomena have distinct, that is, unique identifiers. Being simple entity phenomena, separately observable in the domain, means that their spatial (positional) properties

[24] For concepts one really should speak of conceptual locations, say ℓ:L, rather than "real" locations.
[25] The operator $\mathcal{S}_I$ is meta-linguistic with respect to RSL: it is not part of RSL, but applies to RSL values.
[26] The $\mathcal{S}$ stands for "suppress" and the $\mathcal{I}$ for the suppressed unique identifier.

are distinct. That is their anonymous values are distinct. Meta-linguistically, that is, going outside the RSL framework[27], we can "formalise" this: s157

type
  E [A models a type of simple entity phenomena]
  I^{28} [I models the type of unique E identifiers]
value
  obs_I: E $\rightarrow I$
axiom
  $\forall$ e,e′:E • obs_I(e)$\neq$obs_I(e′) $\Rightarrow \mathcal{S}_I(e)\neq\mathcal{S}_I$(e′)

s158
The above applies to any kind of observable simple entity *phenomenon* A. It does not necessarily apply to "observable" simple entity *concepts*. **Example:** *Two uniquely identified timetables may have their anonymous values be the exact same value.* s159

Simple entity phenomena, in our ontology, are closely tied to space/time "co-ordinates" — with no two simple entity phenomena sharing overlapping space at any one time. Concepts are, in our ontology, not so constrained, that is, we, conceptually, allow "copies" although uniquely named ! Two seemingly distinct concepts may be the same when "stripped" of their unique names ! s160

• • •

Sections 3, and 4.2–4.3 and the material of Sect. 4.4.1 above, can be summarised by proposing a number of domain description laws. We shall just bring a few of these laws here. Enough, we hope, to spur further research into 'laws of description'.

4.4.2 Space Phenomena Consistency s161

Domain Description Law 1, Space Phenomena Consistency *Two otherwise unique, and hence distinctly observable phenomena can, spatially, not overlap.* □

We can express the *Space/Time Phenomena Consistency Law* meta-linguistically, yet in a proper mathematical manner: s162

type
  E [E is the type name of a class of observable simple entity phenomena]
  I [I is the type name of unique E identifiers]
  $\mathcal{L}$ [$\mathcal{L}$ is the type name of E locations]
value
  obs_I: E $\rightarrow I$
  obs_$\mathcal{L}$: E $\rightarrow \mathcal{L}$
axiom
  $\forall$ e,e′:E • obs_I(e)$\neq$obs_I(e′) $\Rightarrow$ obs_$\mathcal{L}$(e) $\sqcap$ obs_$\mathcal{L}$(e′) $= \emptyset$

We can assume that this law always holds for otherwise unique, and hence distinctly observable phenomena.

4.4.3 Unique Identifiers s163

Domain Description Law 2, Unique Identifiers *If two observable simple entities have the same unique identifier then they are the same simple entity.* □

Any domain description must satisfy this law. The domain describer must, typically through axioms, secure that the domain description satisfy this law. Thus there is a *proof obligation* to be *dispensed*, namely that the unique identifier law holds of a domain description.

[27] but staying within a proper mathematical framework — once we have understood the mathematical properties of $\mathcal{S}_I$ and proper RSL values and 'anonymous' values (which, by the way, are also RSL values)

[28] We have here emphasized I, the type name of the type of unique A identifiers. Elsewhere in this book we treat types of unique identifiers of different types of observable simple entities as "ordinary" RSL types. Perhaps we should have "singled" such unique identifier type names out with a special font ? Well, we'll leave it as is !

4.4.4 Unique Phenomena s164

Domain Description Law 3, Unique Phenomena *If two observable simple entities have different unique identifiers then their values, "stripped" of their unique identifiers are different.* □

Any domain description must satisfy this law. The domain describer must, typically through axioms, secure that the domain description satisfy this law. Thus there is a *proof obligation* to be *dispensed*, namely that the unique phenomena law holds of a domain description.

4.4.5 Space/Time Phenomena Consistency: Monotonicity s165

Domain Description Law 4, Space/Time Phenomena Consistency: Monotonicity *If an entity (that has the location property), at time t is at location ℓ, and at time t' (larger than t) is at location ℓ' (different from ℓ), then it moves monotonically from ℓ to ℓ' during the interval (t, t').* □

s166 Specialisations of this law are, for example, that if the movement is of two simple entities, like two trains, along a single rail track and in the same direction, then where train s_i is in front of train s_j at time t, train s_j cannot be in front of train s_i at time t' (where $t' - t$ is some small time interval).

4.4.6 Discussion s167

There are more laws. And there are most likely laws that have yet to be "discovered" ! Any set of laws must be proven consistent. And any domain description must be proven to adhere to these (and "the" other) laws.

We decided to bring this selection of laws because they are a part of the emerging 'domain science'.

Laws of Sects. 4.4.4–4.4.5 are also mentioned, in some other form, in [85].

5 Discussion s168

5.1 Delineations of 'Philosophy', 'Theory', and 'Science'

5.1.1 What is Philosophy ? s169

Philosophy can be compartmentalised in a number of ways. We shall only consider the following three "parts" of philosophy: Metaphysics (or ontology) is the study of reality. Epistemology is the study of knowledge. Logic is the study of the principles of right reasoning.

Let us examine each of these three in some detail.

s170 **Metaphysics** To us 'Ontology' is the same as 'Metaphysics'. Some of the questions that 'Metaphysics' deals with are: (1) What is ultimate reality? (2) Is it one thing or is it many different things? (3) Can reality be grasped by the senses or is it transcendent? (4) What is the mind and what is its relation to the body? We shall constrain ourselves to consider only aspects of the first three.

s171 **Epistemology** Among the questions that 'Epistemology' deals with are: (1) What is knowledge? (2) Is knowledge acquired exclusively through the senses or by some other means? (3) How do we know that what we perceive through our senses is correct?

s172 **Logic** 'Logic' is the basic tool that philosophers use to investigate reality. Logic is the study of the principles of right reasoning. Among the questions raised by Logic are: (1) What is truth. (2) What makes an argument valid or invalid. (3) What is a sound argument?

5.1.2 Natural Sciences Versus Informatics s173

The natural sciences differ fundamentally from those of informatics: In natural sciences we observe what can be measured. Empirical denotes information gained by means of observation, experience, or experiment. A central concept in science and the scientific method is that all evidence must be empirical, or empirically based, that is, dependent on evidence or consequences that are observable by the senses. In informatics we construct our artifacts such that they "fit within computer & communication". These artifacts: data and programs, obey laws of mathematics and should also obey laws of the domains in which they serve, s174

So when we try outline a theory, then it is not a natural sciences theory, but an informatics theory.

Well, yes, we do observe a reality. But is is not a natural sciences reality. It is the reality of a human-made domain: an air traffic system, a container line industry, a financial service industry or a railway transport system. s175

The phenomena researched by the natural sciences remain stable, i.e., are independent of who researches them (or are they ?). The human domain phenomena researched by domain engineers are more subjective, less stable, may change over time, and thus domain models cannot be used for prediction.

5.1.3 What is a Theory ? s176

Therefore, when we have to look at the term 'theory', we have to recall that a natural science theory is different from a domain (or, broader, an informatics) theory.

Some Conventional Definitions of 'Theory' In natural sciences, a theory is a coherent set of propositions that explain a class of phenomena that are supported by extensive factual evidence and that may be used for prediction of future observations.[29] A natural science theory is a scientific account of phenomena. At a minimum theory is a strategy for handling data in research, providing a conceptual system for describing and explaining. A natural science theory is a set s177 of statements,[30] including some law-like generalizations, systematically and logically related such that the set implies something about reality. It is an argument that purports to provide a necessary and sufficient explanation for a range of phenomena. It must be capable of corrigibility - that is, it must be possible to dis-confirm or jeopardize it by making observations. A theory is valuable to the extent that it reduces the uncertainty about the outcome of a specific set of conditions.

An Informatics Theory s178

5.1.4 "The" Scientific Method in Natural Sciences s179

Conventionally scientists pursue their science inquieries along basically the following lines:

1. **Existing theories:** Pose the question in the context of existing knowledge (theory & observations). (It can be a new question that old theories are capable of answering (usually the case), or the question that calls for formulation of a new preliminary theory.) Go to 2.
2. **Hypothesis:** Formulate a hypothesis as a tentative answer. Go to 3.
3. **Predictions and observations:** Deduce consequences and make predictions. Go to 4.
4. **Tests and new observations:** Test the hypothesis in a specific new experiment/theory field. The new hypothesis must prove to fit in the existing world-view (1, "normal science", according to Kuhn). In case the hypothesis leads to contradictions and demands a radical change in the existing theoretical background, it has to be tested particularly carefully. The new hypothesis has to prove fruitful and offer considerable advantages, in order to replace

[29] John A. Cagle, Theories of Human Communication, Littlejohn & Foss Publ.
[30] Darnell

the existing scientific paradigm. This is called "scientific revolution" (Kuhn) and it happens very rarely. Repeat stages 2–3–4 with modified hypothesis until agreement is obtained. This leads to stage 5. Else start from stage 1.

5. **Old theory confirmed or new theory proposed:** When consistency is obtained the hypothesis becomes a theory and provides a coherent set of propositions that define a new class of phenomena or a new theoretical concept. A theory is then becoming a framework within which observations/theoretical facts are explained and predictions are made. The process can start from the beginning, but stage 1 has changed to include the new theory/improved old theory.
6. **Selection among competing theories:** Theory at that stage is subject of process of natural selection among competing theories.

5.2 What is A Philosophy of Informatics ? s180

5.2.1 Ontology of Informatics s181

Some issues of meta-physics are: (1) What is ultimate reality? (2) Is it one thing or is it many different things? (3) Can reality be grasped by the senses or is it transcendent? s182

(1) **What is ultimate reality?**

MORE TO COME

s183

(2) **Is reality one thing or is it many different things?**

MORE TO COME

s184

(3) **Can reality be grasped by the senses or is it transcendent?**

MORE TO COME

5.2.2 Epistemology of Informatics s185

Some issues of epistemology are: (1) What is knowledge? (2) Is knowledge acquired exclusively through the senses or by some other means? (3) How do we know that what we perceive through our senses is correct? s186

(1) **What is knowledge?**

MORE TO COME

s187

(2) **Is knowledge acquired exclusively through the senses or by some other means?**

MORE TO COME

s188

(3) **How do we know that what we perceive through our senses is correct?**

MORE TO COME

5.2.3 Wider Perspectives s189

Topics of study within a philosophy of informatics would most likely start in the sub-fields of computer and computing science and might include such topics as: *what are the sources of computer science subject matter*, *is Church's Thesis adequate for the understanding of computing*, cf. [34], *what is the ontological status of domain entities*, *what is the rôle of hermeneutics in computer science*, *what kinds of inquiry play a rôle in computer science*, *what are the objectives of computer science inquiry*, *what gives computer science its hold on experience*, *what are the human traits behind computer science*, *what is computer science beauty*, *what is the relationship between the abstract world of computer science and the material universe*, and *how do we know whether a computer science proof is correct?*

5.3 What is a Theory of The Science of Informatics ? s190

6 Closing s191

This end my sketches, musings and ruminations. From a characterisation of the "sub-fields", the 'disciplines', of informatics, viz.: computer science, computing sciences and software engineering being some such fields, we moved on to briefly sketch software engineering aspects of domain engineering, requirements engineering, and software design. With the above topics as background we s192 could then focus on a description ontology of simple entities, actions, events and behaviours, and two facets with a[n even more] philosophical bent: Bertrand Russell's Philosophy of Logical Atomism with its clear relations to "our" description ontology, and Stanisław Leśnieksi's mereology with its focus on atomic versus composite parts and and their relations. There are many aspects s193 we have not covered: atomic "versus" composite actions, events and behaviours; description laws, uniqueness of descriptions: *when can we claim that a description is normative ?* et cetera; And, of course, we really have not said anything about what would constitute a *Philosophy of Informatics*, nor what would constitute a *Theory of a Science Informatics*. But hopefully there are some "bits s194 & pieces", maybe even some "nuggets" to be drawn from this talk ? Thanks for your patience ! Any questions ?

7 Bibliographical Notes

The present draft paper is the fourth in our attempt to analyse what can be described and how such descriptions structure their presentation of domains [19, 16, 14].

References

[1] Jean-Raymond Abrial. *The B Book: Assigning Programs to Meanings*. Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, England, 1996.

[2] Jean-Raymond Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, Cambridge, England, 2009.

[3] Aristotle. *Metaphysics, Books I-IX*. Loeb Classical Library No. 271. Harvard University Press, Cambridge, Mass., USA, 1933 (1996).

[4] Peter Bernus and Laszlo 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.

[5] Michel Bidoit and Peter D. Mosses. CASL *User Manual*. LNCS 2900 (IFIP Series). Springer, 2004. With chapters by T. Mossakowski, D. Sannella, and A. Tarlecki.

[6] Dines Bjørner. Domain Models of "The Market" — in Preparation for E–Transaction Systems. In *Practical Foundations of Business and System Specifications (Eds.: Haim Kilov and Ken Baclawski)*, The Netherlands, December 2002. Kluwer Academic Press.

[7] Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[8] Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen.

[9] Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[10] Dines Bjørner. A Container Line Industry Domain. Techn. report, http://www2.imm.dtu.dk/˜db/container-paper.pdf, Fredsvej 11, DK-2840 Holte, Denmark, June 2007.

[11] Dines Bjørner. A Financial Services Industry Domain. Techn. report, http://www2.imm.dtu.dk/˜db/fsi.pdf, Fredsvej 11, DK-2840 Holte, Denmark, 2007.

[12] Dines 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.

[13] Dines Bjørner. A Domain Model of Oil Pipelines. Techn. report, http://www2.imm.dtu.dk/˜db/pipeline.pdf, Fredsvej 11, DK-2840 Holte, Denmark, June 2009.

[14] Dines Bjørner. An Emerging Domain Science – A Rôle for Stanisław Leśniewski's Mereology and Bertrand Russell's Philosophy of Logical Atomism. *Higher-order and Symbolic Computation*, 2009.

[15] Dines 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, 2009. Springer.

[16] Dines Bjørner. On Mereologies in Computing Science. In *Festschrift for Tony Hoare*, History of Computing (ed. Bill Roscoe), London, UK, 2009. Springer.

[17] Dines Bjørner. What is Logistics ? A Domain Analysis. Techn. report, http://www2.imm.dtu.dk/˜db/pipeline.pdf, Fredsvej 11, DK-2840 Holte, Denmark, June 2009.

[18] Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*. JAIST Press, March 2009. JAIST Research Monograph #4, 536 pages: http://www2.imm.dtu.dk/˜db/jaistmono.pdf.

[19] Dines Bjørner and Asger Eir. *Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in Festschrift for Prof. Willem Paul de Roever, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann*. Lecture Notes in Computer Science. Springer, Heidelberg, July 2008.

[20] Dines Bjørner and Martin C. Henson, editors. *Logics of Specification Languages*. EATCS Series, Monograph in Theoretical Computer Science. Springer, Heidelberg, Germany, 2008.

[21] Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer–Verlag, 1978.

[22] Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.

[23] Dines Bjørner and Martin Pěnička. Towards a TRAIN Book for The RAIlway DomaiN. Techn. reports, http://www.railwaydomain.org/PDF/tb.pdf, The TRAIN Consortium, 2004.

[24] Wayne D. Blizard. A Formal Theory of Objects, Space and Time. *The Journal of Symbolic Logic*, 55(1):74–89, March 1990.

[25] Dominique Cansell and Dominique Méry. *Logics of Specification Languages*, chapter The event-B Modelling Method: Concepts and Case Studies, pages 47–152 in [20]. Springer, 2008.

[26] Rudolf Carnap. *Der Logische Aufbau der Welt.* Weltkreis, Berlin, 1928.

[27] R. Casati and A. Varzi. *Parts and Places: the structures of spatial representation.* MIT Press, 1999.

[28] W.J. Clancey. The knowledge–level reinterpreted: modeling socio–technical systems. *International Journal of Intelligent Systems*, 8:33–49, 1993.

[29] Bowman L. Clarke. A Calculus of Individuals Based on 'Connection'. *Notre Dame J. Formal Logic*, 22(3):204–218, 1981.

[30] Bowman L. Clarke. Individuals and Points. *Notre Dame J. Formal Logic*, 26(1):61–75, 1985.

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

[32] CoFI (The Common Framework Initiative). CASL *Reference Manual*, volume 2960 of *Lecture Notes in Computer Science (IFIP Series).* Springer–Verlag, 2004.

[33] Werner Damm and David Harel. LSCs: Breathing life into Message Sequence Charts. *Formal Methods in System Design*, 19:45–80, 2001. Early version appeared as Weizmann Institute Tech. Report CS98-09, April 1998. An abridged version appeared in *Proc. 3rd IFIP Int. Conf. on Formal Methods for Open Object-based Distributed Systems* (FMOODS'99), Kluwer, 1999, pp. 293–312.

[34] Nachum Dershowitz and Yuri Gurevich. A Natural Axiomatization of Computability and Proof of Church's Thesis. *The Bulletin of Symbolic Logic*, 14(3):299–350, September 2008.

[35] Răzvan Diaconescu. *Logics of Specification Languages*, chapter A Methodological Guide to the CafeOBJ Logic, pages 153–240 in [20]. Springer, 2008.

[36] David John Farmer. *Being in time: The nature of time in light of McTaggart's paradox.* University Press of America, Lanham, Maryland, 1990. 223 pages.

[37] Jørgen Fischer Nilsson. *On Reducing Relationships to Property Ascriptions*, volume 190 of *Frontiers in Artificial Intelligence and Applications*, chapter Information Modelling and Knowledge Bases #XX. IOS Press, Amsterdam, The Netherlands, eds. Y. Kiyoki, T. Tokuda, H. Jaakkola, X. Chen and N. Yoshida, January 2009.

[38] John S. Fitzgerald. *Logics of Specification Languages*, chapter The Typed Logic of Partial Functions and the Vienna Development Method, pages 453–487 in [20]. Springer, 2008.

[39] John S. Fitzgerald and Peter Gorm Larsen. *Developing Software using* `VDM-SL`. Cambridge University Press, The Edinburgh Building, Cambridge CB2 1RU, England, 1997.

[40] K. Futatsugi, A.T. 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.

[41] Chris George and Anne E. Haxthausen. *Logics of Specification Languages*, chapter The Logic of the RAISE Specification Language, pages 349–399 in [20]. Springer, 2008.

[42] Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Method.* The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.

[43] Nelson Goodman. *The Structure of Appearance.* 1st.: Cambridge, Mass., Harvard University Press; 2nd.: Indianapolis, Bobbs-Merrill, 1966; 3rd.: Dordrecht, Reidel, 1977, 1951.

[44] Nelson Goodman. *A Study of Qualities.* Garland, New York, 1990. Ph.D. dissertation thesis, Harvard, 1940.

[45] Thomas R. Gruber and Gregory R. Olsen. An Ontology for Engineering Mathematics. In Jon Doyle, Piero Torasso, and Erik Sandewall, editors, *Principles of Knowledge Representation and Reasoning.* Morgan Kaufmann, 1994. Fourth International Conference. Gustav Stresemann Institut, Bonn, Germany.[31].

[46] M. Gruninger and M.S. Fox. The Logic of Enterprise Modelling. In *Modelling and Methodologies for Enterprise Integration, see [4]*, pages 141–157, November 1995.

[47] Nicola Guarino. Formal Ontology, Conceptual Analysis and Knowledge Representation. *Intl. Journal of Human–Computer Studies*, 43:625–640, 1995.

[48] Nicola Guarino. Some Organising Principles for a Unified Top–level Ontology. Int.rept., Italian National Research Council (CNR), LADSEB–CNR, Corso Stati Uniti 4, I–35127 Padova, Italy. guarino@ladseb.pd.cnr.it, 1997.

[49] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[50] David Harel and Rami Marelly. *Come, Let's Play – Scenario-Based Programming Using LSCs and the Play-Engine.* Springer-Verlag, 2003.

[51] Martin C. Henson, Moshe Deutsch, and Steve Reeves. *Logics of Specification Languages*, chapter Z Logic and Its Applications, pages 489–596 in [20]. Springer, 2008.

[52] Tony Hoare. *Communicating Sequential Processes.* C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.

[53] Tony Hoare. Communicating Sequential Processes. Published electronically: `http://www.-usingcsp.com/cspbook.pdf`, 2004. Second edition of [52]. See also `http://www.usingcsp.-com/`.

[54] ITU-T. CCITT Recommendation Z.120: Message Sequence Chart (MSC), 1992.

[55] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1996.

[56] ITU-T. ITU-T Recommendation Z.120: Message Sequence Chart (MSC), 1999.

[57] Daniel Jackson. *Software Abstractions Logic, Language, and Analysis.* The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.

[58] Imre Lakatos. *Proofs and Refutations: The Logic of Mathematical Discovery (Eds.: J. Worrall and E. G. Zahar).* Cambridge University Press, The Edinburgh Building, Shaftesbury Road, Cambridge CB2 2RU, England, 2 September 1976. ISBN: 0521290384. Published in 1963-64 in four parts in the British Journal for Philosophy of Science. (Originally Lakatos' name was Imre Lipschitz.).

[59] Leslie Lamport. *Specifying Systems.* Addison–Wesley, Boston, Mass., USA, 2002.

[60] Henry S. Leonard and Nelson Goodman. The Calculus of Individuals and its Uses. *Journal of Symbolic Logic*, 5:45–44, 1940.

[61] Stanisław Leśniewksi. 0 Podstawack Matematyki (Foundations of Mathematics). *Prezeglad Filosoficzny*, 30-34, 1927-1931.

[62] E.C. Luschei. *The Logical Systems of Leśniewksi.* North Holland, Amsterdam, The Netherlands, 1962.

[31] `http://www-ksl.stanford.edu/knowledge-sharing/papers/engmath.html`

[63] John McCarthy. Towards a Mathematical Science of Computation. In C.M. Popplewell, editor, *IFIP World Congress Proceedings*, pages 21–28, 1962.

[64] J. M. E. McTaggart. The Unreality of Time. *Mind*, 18(68):457–84, October 1908. New Series. See also: [67].

[65] Stephan Merz. *Logics of Specification Languages*, chapter The Specification Language TLA$^+$, pages 401–451 in [20]. Springer, 2008.

[66] T. Mossakowski, A. Haxthausen, D. Sannella, and A. Tarlecki. *Logics of Specification Languages*, chapter CASL – the Common Algebraic Specification Language, pages 241–298 in [20]. Springer, 2008.

[67] Robin Le Poidevin and Murray MacBeath, editors. *The Philosophy of Time*. Oxford University Press, 1993.

[68] Karl R. Popper. *Logik der Forschung*. Julius Springer Verlag, Vienna, Austria, 1934 (1935). English version [69].

[69] Karl R. Popper. *The Logic of Scientific Dicovery*. Hutchinson of London, 3 Fitzroy Square, London W1, England, 1959,...,1979. Translated from [68].

[70] Karl R. Popper. *Conjectures and Refutations. The Growth of Scientific Knowledge*. Routledge and Kegan Paul Ltd. (Basic Books, Inc.), 39 Store Street, WC1E 7DD, London, England (New York, NY, USA), 1963,...,1981.

[71] Karl R. Popper. *Autobiography of Karl Popper*. in The Philosohy of Karl Popper in *The Library of Living Philosophers*, Ed. Paul Arthur Schilpp. Open Court Publishing Co., Illionos, USA, 1976. See also [72].

[72] Karl R. Popper. *Unended Quest: An Intellectual Autobiography*. Philosophy and Autobiography. Fontana/Collins, England, 1976–1982. Originally in [71].

[73] Arthur Prior. *Changes in Events and Changes in Things*, chapter in [67]. Oxford University Press, 1993.

[74] Arthur N. Prior. *Logic and the Basis of Ethics*. Clarendon Press, Oxford, UK, 1949.

[75] Arthur N. Prior. *Time and Modality*. Oxford University Press, Oxford, UK, 1957.

[76] Arthur N. Prior. *Past, Present and Future*. Clarendon Press, Oxford, UK, 1967.

[77] Arthur N. Prior. *Papers on Time and Tense*. Clarendon Press, Oxford, UK, 1968.

[78] Wolfang Reisig. *A Primer in Petri Net Design*. Springer Verlag, March 1992. 120 pages.

[79] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs in Theoretical Computer Science*. Springer Verlag, May 1985.

[80] Wolfgang Reisig. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets*. Springer Verlag, December 1998. xi + 302 pages.

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

[82] A. W. Roscoe. *Theory and Practice of Concurrency*. C.A.R. Hoare Series in Computer Science. Prentice-Hall, 1997. Now available on the net: http://www.comlab.ox.ac.uk/people/-bill.roscoe/publications/68b.pdf.

[83] Bertrand Russel. *"Preface," Our Knowledge of the External World.* G. Allen & Unwin, Ltd., London, 1952.

[84] Bertrand Russell. *The Principles of Mathematics.* Allen and Unwin, London, 1910.

[85] Bertrand Russell. The Philosophy of Logical Atomism. *The Monist: An International Quarterly Journal of General Philosophical Inquiry,*, xxxviii–xxix:495–527, 32–63, 190–222, 345–380, 1918–1919.

[86] Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach.* Worldwide Series in Computer Science. John Wiley & Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, January 2000.

[87] John G. Slater, editor. *The Collected Papers of Bertrand Russel.* Allen and Unwin, London, England, 1986.

[88] Barry Smith. Mereotopology: A Theory of Parts and Boundaries. *Data and Knowledge Engineering*, 20:287–303, 1996.

[89] John F. Sowa. *Knowledge Representation: Logical, Philosophical, and Computational Foundations.* Pws Pub Co, August 17, 1999. ISBN: 0534949657, 512 pages, Amazon price: US $ 70.95.

[90] J.T.J. Srzednicki and Z. Stachniak, editors. *Leśniewksi's Lecture Notes in Logic.* Dordrecht, 1988.

[91] Steffen Staab and Rudi Stuber, editors. *Handbook on Ontologies.* International Handbooks on Information Systems. Springer, Heidelberg, 2004.

[92] A. Tarski. , chapter Appendix E. Cambridge University Press, 1937.

[93] A. Tarski. *Foundations of the geometry of solids.* Oxford University Press, Oxford, 1956. Transl. J. H. Woodger,.

[94] Johan 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.

[95] Achille C. Varzi. *On the Boundary between Mereology and Topology*, pages 419–438. Hölder-Pichler-Tempsky, Vienna, 1994.

[96] Achille C. Varzi. *Spatial Reasoning in a Holey*[32] *World*, volume 728 of *Lecture Notes in Artificial Intelligence*, pages 326–336. Springer, 1994.

[97] A.N. Whitehead. *The Concept of Nature.* Cambridge University Press, Cambridge, 1920.

[98] A.N. Whitehead. *An Enquiry Concerning the Principles of Natural Knowledge.* Cambridge University Press, Cambridge, 2929.

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

[100] J.H. Woodger. *The Axiomatic Method in Biology.* Cambridge University Press, Cambridge, 1937.

[101] Chao Chen Zhou and Michael R. Hansen. *Duration Calculus: A Formal Approach to Real–time Systems.* Monographs in Theoretical Computer Science. An EATCS Series. Springer–Verlag, 2004.

[32]holey: something full of holes

[102] Chao Chen Zhou, Charles Anthony Richard Hoare, and Anders P. Ravn. A Calculus of Durations. *Information Proc. Letters*, 40(5), 1992.

A An Example Domain Description: Railway Nets s-10

1. A railway net consists of one or more lines and two or more stations.

```
type
   1. RN, LI, ST
value
   1. obs_LIs: RN → LI-set
   1. obs_STs: RN → ST-set
axiom
   1. ∀ n:RN • card obs_LIs(n) ≥ 1 ∧ card obs_STs(n) ≥ 2
```

s-9

2. A railway net consists of rail units.

```
type
   2. U
value
   2. obs_Us: RN → U-set
```

3. A line is a linear sequence of one or more linear rail units.

```
axiom
   3. ∀ n:RN, l:LI • l ∈ obs_LIs(n) ⇒ lin_seq(l)
```

s-8

4. The rail units of a line must be rail units of the railway net of the line.

```
value
   3. obs_Us: LI → U-set
axiom
   4. ∀ n:RN, l:LI • l ∈ obs_LIs(n) ⇒ obs_Us(l) ⊆ obs_Us(n)
```

5. A station is a set of one or more rail units.

```
value
   5. obs_Us: ST → U-set
axiom
   5. ∀ n:RN, s:ST • s ∈ obs_STs(n) ⇒ card obs_Us(s) ≥ 1
```

s-7

6. The rail units of a station must be rail units of the railway net of the station.

```
axiom
   6. ∀ n:RN, s:ST • s ∈ obs_STs(n) ⇒ obs_Us(s) ⊆ obs_Us(n)
```

7. No two distinct lines and/or stations of a railway net share rail units.

```
axiom
   7. ∀ n:RN,l,l′:LI•{l,l′}⊆obs_LIs(n)∧l≠l′⇒obs_Us(l)∩ obs_Us(l′)={}
   7. ∀ n:RN,l:LI,s:ST•l ∈ obs_LIs(n)∧s ∈ obs_STs(n)⇒obs_Us(l)∩ obs_Us(s)={}
   7. ∀ n:RN,s,s′:ST•{s,s′}⊆obs_STs(n)∧s≠s′⇒obs_Us(s)∩ obs_Us(s′)={}
```

s-6

8. A station consists of one or more tracks.

```
type
   8. Tr
value
   8. obs_Trs: ST → Tr-set
axiom
   8. ∀ s:ST•card obs_Trs(s)≥1
```

9. A track is a linear sequence of one or more linear rail units.

```
axiom
    9. $\forall$ n:RN,s:ST,t:Tr•s $\in$ obs_STs(n)$\wedge$t $\in$ obs_Trs(s)$\Rightarrow$lin_seq(t)
```

s-5

10. No two distinct tracks share rail units.

```
axiom
    10. $\forall$ n:RN,s:ST,t,t$'$:Tr•s $\in$ obs_STs(n)$\wedge$$\{$t,t$'\}$$\subseteq$obs_Trs(s)$\wedge$t$\neq$t$'$$\Rightarrow$obs_Us(t) $\cap$ obs_Us(t$'$)=$\{\}$
```

11. The rail units of a track must be rail units of the station (of that track).

```
value
    11. obs_Us: Tr $\rightarrow$ U-set
axiom
    11. $\forall$ rn:RN,st:ST,tr:TR •
                st $\in$ obs_STs(rn)$\wedge$tr $\in$ obs_Trs(st)$\Rightarrow$obs_Us(tr)$\subseteq$obs_Us(st)
```

s-4

12. A rail unit is either a linear, or is a switch, or is a simple crossover, or is a switchable crossover, etc., rail unit.

```
value
    12. is_Linear: U $\rightarrow$ Bool
    12. is_Switch: U $\rightarrow$ Bool
    12. is_Simple_Crossover: U $\rightarrow$ Bool
    12. is_Switchable_Crossover: U $\rightarrow$ Bool
```

13. A rail unit has one or more connectors.

```
type
    13. K
value
    13. obs_Ks: U $\rightarrow$ K-set
```

s-3

14. A linear rail unit has two distinct connectors. A switch (a point) rail unit has three distinct connectors. Crossover rail units have four distinct connectors (whether simple or switchable), etc.

```
axiom
    $\forall$ u:U •
        is_Linear(u) $\Rightarrow$ card obs_Ks(u)=2$\wedge$
        is_Switch(u) $\Rightarrow$ card obs_Ks(u)=3$\wedge$
        is_Simple_Crossover(u) $\Rightarrow$ card obs_Ks(u)=4$\wedge$
        is_Switchable_Crossover(u) $\Rightarrow$ card obs_Ks(u)=4
```

15. For every connector there are at most two rail units which have that connector in common.

```
axiom
    15. $\forall$ n:RN • $\forall$ k:K • k $\in$ $\cup$$\{$obs_Ks(u)|u:U•u $\in$ obs_Us(n)$\}$
            $\Rightarrow$card$\{$u|u:U•u $\in$ obs_Us(n)$\wedge$k $\in$ obs_Ks(u)$\}$$\leq$2
```

s-2

16. Every line of a railway net is connected to exactly two distinct stations of that railway net.

```
axiom
    16. $\forall$ n:RN,l:LI • l $\in$ obs_LIs(n) $\Rightarrow$
        $\exists$ s,s$'$:ST • $\{$s,s$'\}$ $\subseteq$ obs_STs(n) $\wedge$ s$\neq$s$'$ $\Rightarrow$
            let sus=obs_Us(s),sus$'$=obs_Us(s$'$),lus=obs_Us(l) in
            $\exists$ u,u$'$,u$''$,u$'''$:U • u $\in$ sus $\wedge$
                u$'$ $\in$ sus$'$ $\wedge$ $\{$u$''$,u$'''\}$ $\subseteq$ lus $\Rightarrow$
                let sks = obs_Ks(u), sks$'$ = obs_Ks(u$'$),
                    lks = obs_Ks(u$''$), lks$'$ = obs_Ks(u$'''$) in
                $\exists$!k,k$'$:K•k$\neq$k$'$$\wedge$sks $\cap$ lks=$\{$k$\}$$\wedge$sks$'$ $\cap$ lks$'$=$\{$k$'\}$
            end end
```

s-1

17. A linear sequence of (linear) rail units is an acyclic sequence of linear units such that neighbouring units share connectors.

```
value
    17.  lin_seq: U-set → Bool
           lin_seq(us) ≡
               ∀ u:U • u ∈ us ⇒ is_Linear(u) ∧
                   ∃ q:U* • len q = card us ∧ elems q = us ∧
                       ∀ i:Nat • {i,i+1} ⊆ inds q ⇒
                           ∃ k:K • obs_Ks(q(i)) ∩ obs_Ks(q(i+1)) = {k} ∧
                           len q > 1 ⇒ obs_Ks(q(i)) ∩ obs_Ks(q(len q)) = {}
```