

---

# Compositionality: Ontology and Mereology of Domains\*

Some Clarifying Observations in the Context of Software Engineering

Dines Bjørner<sup>1</sup> and Asger Eir<sup>2</sup>

<sup>1</sup> DTU Informatics, Techn. Univ. of Denmark, DK-2800 Kgs. Lyngby, Denmark\*\*

<sup>2</sup> Maconomy, Vordingborggade 18-22, DK-2100 Copenhagen Ø, Denmark\*\*\*

**Summary.** In this discursive paper we discuss compositionality of (i) simple entities, (ii) operations, (iii) events and (iv) behaviours. These four concepts, (i)–(iv), together define a concept of entities. We view entities as “things” characterised by properties. We shall review some such properties. Mereology, the study of part-whole relations is then applied to a study of composite entities. We then speculate on compositionality of simple entities, operations, events and behaviours in the light of their mereologies. entities. We end the paper with some speculations on the rôle of Galois connections in the study of compositionality and domain mereology.

## 1 A Prologue Example

We begin with an example: an informal and formal description of fragments of a domain of transportation. The purpose of such an example is to attach this example to our discussion of entities, and to enlarge the example with further examples to support this discussion of entities, and hence of mereology and ontology. The formalisation of the example narratives is expressed in the RAISE Specification Language, RSL [30, 32, 6, 7, 8, 31, 29, 13] — but could as well have been expressed in Alloy, ASM, Event B, VDM or Z [43, 58, 59, 1, 18, 15, 16, 26, 25, 64, 65, 68, 36, 35].

**Narrative:** (0.) There are links and there are hubs, (1.) Links and hubs have unique identifiers. (2.) Transport net consists of links and hubs. We can either model nets as sorts and then observe links and hubs from nets:

---

\* Invited paper for the Willem-Paul de Roever Festschrift, July 2008; eds.: Martin Steffen, Dennis Dams and Ulrich Hannemann. This paper is a full draft version of the paper. It should appear in an pre-publication hand-out at that Festschrift. A final version is expected to appear in a Springer Festschrift volume later in 2008. It is hoped that that published paper will be a shortened version of the present.

\*\* Home address: Fredsvej 11, DK-2840 Holte, Denmark. Professor Emeritus. E-mail: [bjorner@gmail.com](mailto:bjorner@gmail.com)

\*\*\* E-mail: [aei@maconomy.dk](mailto:aei@maconomy.dk), [asger@eir-home.dk](mailto:asger@eir-home.dk), URL: [www.eir-home.dk](http://www.eir-home.dk)

|  |  |
|--|--|
| <b>type</b><br>N, L, H,<br><b>value</b><br>obs_Ls: N $\rightarrow$ L-set,<br>obs_Hs: N $\rightarrow$ H-set | or<br><br><b>type</b><br>L, H,<br>N = L-set $\times$ H-set |
|--|--|

(3.) Links connect exactly two distinct hubs. (4.) Hubs are connected to one or more distinct links. (5.) From a link one can observe the two unique identifiers of the hubs to which it is connected. (6.) From a hub one can observe the set of one or more unique identifiers of the links to which it is connected. (7.) Observed unique link (hub) identifiers are indeed identifiers of links (hubs) of the net in which the observation takes place.

**Formalisation:**

**type**  
 0.–1. L, LI, H, HI,  
 2. N = L-set  $\times$  H-set  
**axiom**  
 3.–4.  $\forall (ls,hs):N \bullet \mathbf{card} \text{ } ls \geq 1 \wedge \mathbf{card} \text{ } hs \geq 2$   
**value**  
 1. obs\_LI: L  $\rightarrow$  LI, obs\_HI: H  $\rightarrow$  HI,  
 5. obs\_HIs: L  $\rightarrow$  HI-set **axiom**  $\forall l:L \bullet \mathbf{card} \text{ } obs\_HIs(l)=2,$   
 6. obs\_LIs: H  $\rightarrow$  LI-set **axiom**  $\forall h:H \bullet \mathbf{card} \text{ } obs\_LIs(l) \geq 1$   
**axiom**  
 7.  $\forall (ls,hs):N \bullet$   
      $\forall l:L \bullet l \in ls \Rightarrow$   
          $\forall hi:HI \bullet hi \in obs\_HIs(l) \Rightarrow \exists h:H \bullet hi=obs\_HI(h) \wedge h \in hs$   
      $\wedge \forall h:H \bullet h \in hs \Rightarrow$   
          $\forall li:LI \bullet li \in obs\_LIs(l) \Rightarrow \exists l:L \bullet li=obs\_LI(k) \wedge l \in ls$

**Narrative:** (8.) There are vehicles (private cars, taxis, buses, trucks). (9.) Vehicles, when “on the net”, i.e., “in the traffic” (see further on), have positions. Vehicle positions are (10.) either at a hub, in which case we could speak of the hub identifier as being a suitable designation of its location, (11.) or along a link, in which case we could speak of of a quadruple of a (from) hub identifier, a(n along) link identifier, a real (a fraction) properly between 0 and 1 as designating a relative displacement “down” the link, and a (to) hub identifier, as being a suitable designation of its location, (12.) Time is a discrete, dense well-ordered set of time points and time points are further undefined. (13.) Traffic can be thought of as a continuous function from time to vehicle positions. We augment our model of traffic with the net “on which it runs”!

**Formalisation:**

**type**  
 8. V

9.  $VPos == HubPos \mid LnkPos$
10.  $HubPos = HP(hi:HI)$
11.  $LnkPos = LP(fhi:HI,li:LI,f:Real,thi:HI)$
12. Time
13.  $TRF = (Time \rightarrow (V \xrightarrow{m} VPos)) \times N$

**Closing Remarks:** We omit treatment here of traffic well-formedness: that time changes and vehicle movement occurs monotonically; that there are no “ghost” vehicles (vehicles “disappear” only to “reappear”), that two or more vehicles “one right after the other” do not “suddenly” change relative positions while continuing to move in the same direction, etc.

## 2 Introduction

The narrow context of this essay is that of domain engineering: the principles, techniques and tools for describing domains, as they are, with no consideration of software, hence also with no consideration of requirements. The example of Sect. 1 describes (narrates and formalises) some aspects of a domain.

The broader context of this essay is that of formal software engineering: the phase, stage and stepwise development of software, starting with *Domain* descriptions, evolving into *Requirements* prescriptions and ending with *Software* design in such a way that  $\mathcal{D}, \mathcal{S} \models \mathcal{R}$ , that is: software can be proven correct with respect to requirements with the proofs and the correctness relying on the domain as described.

### 2.1 Domain Engineering

**The Domain Engineering Dogma:** Before software be designed, we must understand its requirements. Before requirements can be expressed, we must understand the application domain.

**The Software Development Triptych:** Thus, we must first describe the domain as it is. Then we can prescribe the requirements as we would like to see them implemented in software. First then can we specify the design of that software.

**Domain Descriptions:** A domain description specifies the domain **as it is**. (The example traffic thus allows vehicles to crash.) A domain description does not hint at requirements let alone software to be designed. A domain description specifies observable domain phenomena and concepts derived from these.

**Example:** *a vehicle is a phenomenon; a vehicle position is also a phenomenon, but the way in which we suggest to model a position is a concept; similarly for traffic* ■

A domain description does not describe human sentiments (**Example:** *the bus ride is beautiful* ■), opinions and thoughts (**Example:** *the bus ride is a bit too expensive* ■), knowledge and belief (**Example:** *I know of more beautiful rides* ■ and *I believe there are cheaper bus fares* ■), promise and commitment (**Example:** *I promise to take you on a more beautiful ride one day* ■) or other such sentential, modal structures.

A domain description primarily specifies semantic entities of the domain intrinsics (**Example:** *the net, links and hubs are semantics quantities* ■), semantic entities of support technologies already “in” the domain, semantic entities of management and organisation domain entities, syntactic and semantic of domain rules and regulations, syntactic and semantic of domain scripts (**Example:** *bus time tables respectively the bus traffic* ■) and semantic aspects of human domain behaviour.

The domain description, to us, is (best) expressed when both informally narrated and formally specified. A problem, therefore, is: can we formalise all the observable phenomena and concepts derived from these? If they are observable or derived, we should be able to formalise. But computing science may not have developed all the necessary formal description tools. We shall comment on that problem as we go along.

## 2.2 Compositionality

We shall view compositionality “in isolation”! That is, not as in the conventional literature where **the principle of compositionality** is the principle that the meaning of a complex expression is determined by the meanings of its constituent expressions and the rules used to combine them. We shall look at not only composite simple entities but also composite operations, events and behaviours in isolation from their meaning but shall then apply the principle of compositionality such that the meaning of a composite operation [event, behaviour] is determined by the meanings of its constituent operations [event, behaviours] and the *rules used for combining* these. We shall, in this paper only go halfway towards this goal: we look only at possible *rules used to combine* simple entities, functions, events and behaviours.

For simple entities we can say the following about compositionality. A key idea seems to be that compositionality requires the existence of a homomorphism between the entities of a universe  $\mathcal{A}$  and the entities in some other universe  $\mathcal{B}$ .

Let us think of the entities of one system,  $\mathcal{A}$ , as a set,  $\mathcal{U}$ , upon which a number of operations are defined. This gives us an algebra  $\mathcal{A} = (\mathcal{U}, \mathcal{F}_\nu)_{\nu \in \Gamma}$  where  $\mathcal{U}$  is the set of (simple and complex) entities and every  $\mathcal{F}_\nu$  is an operation on  $\mathcal{A}$  with a fixed arity. The algebra  $\mathcal{A}$  is interpreted through a meaning-assignment  $\mathcal{M}$ ; a function from  $\mathcal{U}$  to  $\mathcal{V}$ , the set of available meanings for the entities of  $\mathcal{U}$ . Now consider  $\mathcal{F}_\nu$ ; a  $k$ -ary syntactic operation on  $\mathcal{A}$ .  $\mathcal{M}$  is  $\mathcal{F}_\nu$ -compositional just in case there is a  $k$ -ary function  $\mathcal{G}$  on  $\mathcal{V}$  such that whenever  $\mathcal{F}_\nu(u_1, \dots, u_k)$  is defined

$$\mathcal{F}_\nu(u_1, \dots, u_k) = \mathcal{G}(\mathcal{M}(u_1), \dots, \mathcal{M}(u_k)).$$

In denotational semantics we take this homomorphism for granted, while applying to, as we shall call them, syntactic terms of entities. We shall, in this paper, speculate on compositionality of non-simple entities. That is, compositionality of operations, events and behaviours; that is, of interpretations over non-simple entities (as well as over simple entities).

### 2.3 Ontology

By an ontology we shall understand *an explicit, formal specification of a shared conceptualisation*<sup>3</sup>.

We shall claim that domain engineering, as treated in [6, 9, 11], amounts to principles, techniques and tools for formal specification of shared conceptualisations. The conceptualisation is of a domain, typically a business, an industry or a service domain.

One thing is to describe a domain, that is, to present an ontology for that domain. Another thing is for the description to be anchored around a description ontology: a set of principles, techniques and tools for structuring descriptions. In a sense we could refer to this latter as a meta-ontology, but we shall avoid the prefix ‘meta-’ and instead understand it so. The conceptualisation is of the domain of software engineering methodology, especially of how to describe domains.

### 2.4 Mereology

Mereology is the theory of parthood relations: of the relations of part to whole and the relations of part to part within a whole.

The issue is not simply whether an entity is a proper part,  $p_p$ , of another part,  $p_\omega$  (for example, “the whole”), but also whether a part,  $p_i$ , which is a proper part of  $p_p$  can also be a part of another part,  $p_\xi$  which is not a part of  $p_p$ , etcetera. To straighten out such issues, axiom systems for mereology (part/whole relations) have been proposed [46, 19, 20]. See Appendix A.1.

The term mereology seems to have been first used in the sense we are using it by the Polish mathematical logician Stanisław Leśniewski [49, 66].

The concept of Calculus of Individuals [47, 20, Leonard & Goodman (1940) and Clarke (1981)] is related to that of Mereology. See Appendix A.2.

We shall return to the issue of mereology much more in this paper. In fact, we shall outline “precisely” what our entity mereologies are.

### 2.5 Paper Outline

The paper is structured as follows: after Sect. 2’s brief characteristics of domain engineering, compositionality, ontology and mereology, Sect. 3 overviews

<sup>3</sup> <http://www-ksl.stanford.edu/kst/what-is-an-ontology.html>

what we shall call an ontological aspect of description structures, namely that of entities (having properties). Sections 4–7 will then study (i) simple, (ii) operation, (iii) event and (iv) behaviour entities in more detail, both atomic and composite. For the composite entities we shall then speculate on their mereology. Section 8 concludes our study of some mereological aspects of composite entities by relating these to definitions and axioms of proposed axiom systems for mereology, cf. Appendix A. Section 9 takes a brief look at rôles that the concept of Galois Connections may have in connection with composite entities.

### 3 An Ontology Aspect of Description Structures

This section provides a brief summary of Sects. 4–7.

The choice of analysing a concept of compositionality from the point of view of simple entities, operations, events and behaviours reflects an ontological choice, that is a choice of how we wish to structure our study of conceptions of reality and the nature of being.

We shall take the view that an ontology for the domain of descriptions evolves around the concepts of entities inseparably from their properties. More concretely, “our” ontology consists of entities of the four kinds of specification types: simple entities, operations, events and behaviours. One set of properties is that of an entity being ‘simple’, being ‘an operation’ (or function), being ‘an event’ or being ‘a behaviour’. We shall later introduce further categories of entity properties.

#### 3.1 Simple Entities<sup>4</sup>

In a rather concrete, “mechanistic” sense, we understand simple entities as follows: simple entities have properties which we model as types and values. When a simple entity is concretely represented, “inside” a computer, it is usually implemented in the form of data.

By a **state**,  $\sigma:\Sigma$ , we shall understand a designated set of entities.

Entities are the target of operations: function being applied to entities and resulting in entities.

In Sect. 4 we shall develop this view further.

**Examples:** *The nets, links, hubs, vehicles and vehicle positions of our guiding example are simple entities* ■

Simple domain entities are either atomic or composite. Composite entities are here thought of (i.e., modelled as) finite or infinite sets of (simple) entities:

---

<sup>4</sup> The term ‘simple entity’ is chosen in contrast to the (‘complex’) function, event and behaviour entities. We shall otherwise not use the term ‘complex’ as it has no relation to composition, but may be confused with it.

$\{e_1, e_2, \dots, e_n\}$ , finite Cartesians (i.e., groupings [records, structures] of (simple) entities):  $(e_1, e_2, \dots, e_n)$ , finite or infinite lists (i.e., ordered sequences of (simple) entities):  $\langle e_1, e_2, \dots, e_n \rangle$ , maps (i.e., finite associations of (simple) entities to (simple) entities):  $[e_{d_1} \mapsto e_{r_1}, e_{d_2} \mapsto e_{r_2}, \dots, e_{d_n} \mapsto e_{r_n}]$ , and functions (from (simple) entities to (simple) entities):  $\lambda v : \mathcal{E}(v)$ .<sup>5</sup>

### 3.2 Operations

To us, an **operation** (synonym for function) is something which **when** applied to an entity or an attribute<sup>6</sup> yields an entity or an attribute.

If an operation  $op$  argument and the resulting entity qualify as states  $(\sigma : \Sigma)$ , then we have a state-changing **action**:  $op : [\dots \times] \Sigma \rightarrow \Sigma$ .

If an operation argument entity qualifies as a state and if the resulting entity can be thought of as a pair of which (exactly) one element qualifies as a state, then we have a value yielding action with a, perhaps, beneficial *side effect*:  $op : [\dots \times] \Sigma \rightarrow (\Sigma \times \text{VAL})$ .

If the operation argument does not qualify as a state then we have a value yielding function with no side effect on the state.

Since entities have types we can talk of the signature of an operation as consisting of the name of the operation, the structure of types of its argument entities, and the type of the resulting entities. We gave two such signatures (for operation  $op$ ) above. (The  $[\dots \times]$  indicate that there could be other arguments than the explicitly named state entity  $\Sigma$ .)

**Example:** *The unique identifier observer functions of our guiding example are operations* ■

They apply to entities and yields entities or attributes:  $\text{obs\_Ls} : \text{N} \rightarrow \text{L-set}$  and  $\text{obs\_Hs} : \text{N} \rightarrow \text{H-set}$  yield entities and  $\text{obs\_LI} : \text{L} \rightarrow \text{LI}$  and  $\text{obs\_HI} : \text{H} \rightarrow \text{HI}$  yield attributes.

**“First Class” Entities:** Before closing this section, Sect. 3.2, we shall “lift” operations, hence actions and functions to be first class entities!

### 3.3 Events

In [45, Lamport] events are the same as executed atomic actions. We shall not really argue with that assumption. In [45, Lamport] events are of interest only in connection with the concept of processes (for which we shall use the term ‘behaviours’). We shall certainly follow that assumption. We wish to reserve the term ‘event’ for such actions which (i) are either somehow shared between two or more behaviours, (ii) or ‘occur’ in just one behaviour. We assume

<sup>5</sup> **Note:** The decorated  $es$  in set, Cartesian, list and map enumerations stand for actual entities whereas the  $v$  in  $\lambda v : \mathcal{E}(v)$  is a syntactic variable and  $\mathcal{E}(v)$  stand for a syntactic expression with a free variable  $v$ .

<sup>6</sup> See Sect. 4.1 for distinction between entity and attribute

an “external”, further undefined behaviour. For both of these two cases we need a way of “labelling” events. We do so by labelling,  $\beta\ell_i$ , behaviours,  $\beta_i$ , that is, ascribing names to behaviours. Let the external behaviour have a distinguished, “own” label (e.g.,  $\beta_x\ell$ ). Now we can label an event by the set of labels of the processes “in” which the event occur. That is, with either two or more labels, or just one. When the external behaviour label  $\beta_x\ell$  is in the set then it shall mean that the event either “originates” outside the behaviours of the other labels, or is “directed” at all those behaviours. We do not, however, wish to impose any direction! Here we wish to remind the reader that “our” behaviours take place “in the domain”, that is, they are not necessarily those of computing processes, unless, of course, the domain is, or (“strongly”) includes that of computing; and “in the domain” we can always speak “globally”, that is: we may postulate properties that may not be computable or even precisely observable, that is: two time stamps may be different even though they are of two actions or events that actually did or do take place simultaneously.

Thus: we are not bothered by clocks, that is, we do not enforce a global clock; we do not have to sort out ordering problems of events, but can leave that to a later analysis of the described domain, recommendably along the lines of [45, Lamport].

**Time and Time Stamps:** Time is some dense set of time points.

A time stamp is just a time designator,  $t$ . Two time stamps are consecutive if they differ by some infinitesimal time difference,  $t_\delta$ . We shall assume the simplifying notion of a “global” clock. For the kind of distributed systems that are treated in [45, Lamport] this may not be acceptable, but for a any actual domain that is not subject to Einsteinian relativity, and most are, it will be OK. Once we get to implementation in terms of actual systems possibly governed by erroneously set clocks one shall have to apply for example [45, Lamport]’s treatment.

**Definition: Event:** To us, an event,  $\mathbb{E} : \{(\beta\ell_1, \sigma_1, P_1, \sigma'_1, \tau_1), (\beta\ell_2, \sigma_2, P_2, \sigma'_2, \tau_2), \dots, (\beta\ell_n, \sigma_n, P_n, \sigma'_n, \tau_n)\}$  involves a set of behaviours,  $\beta_i$ , and is expressed in terms of a set of event designators, quintuplets containing:

- ★<sup>1</sup> a label  $\beta\ell_i$ ,
- ★<sup>2</sup> a before state  $\sigma_i$ ;
- ★<sup>3</sup> a predicate  $P_i$ ;
- ★<sup>4</sup> an after state  $\sigma'_i$ ;
- such that  $P_i(\sigma_i, \sigma'_i)$
- but where it may be the case that  $\sigma_i = \sigma'_i$ ;
- ★<sup>5</sup> and a time stamp  $\tau_i$
- which is either a time  $t_i$
- or a time interval  $[t'_i, t''_i]$
- \* such that  $t''_i - t'_i = \tau_{\delta_i} > 0$
- \* but where  $\tau_{\delta_i}$  is otherwise considered “small” ■



An event,  $\mathbb{E}$ , may change one or more behaviour states, selectively, or may not — in which latter case  $\sigma_i = \sigma'_i$  for some  $i$ .

Thus we do not consider the time(s) when expressing conditions  $P_i$ .

**Definition: Same Event:** We assume two or more distinct behaviours  $\beta_1, \beta_2, \dots, \beta_n$ . Two or more events  $\mathbb{E}_{1_i}, \mathbb{E}_{2_i}$  and  $\mathbb{E}_{n_i}$  are said to reflect, i.e., to be the same event iff their models, as suggested above, are ‘identical’ modulo predicates<sup>7</sup> and time stamps, iff these time stamps differ at most “insignificantly”, a decision made by the domain describer<sup>8</sup>, and iff this model involves the label sets  $\beta l_1, \beta l_2, \dots, \beta l_n$  for behaviours  $\beta_1, \beta_2, \dots, \beta_n$  ■

This means that any one event which is assumed to be the same and thus to occur more-or-less simultaneously in several behaviours is “identically” recorded (modulo predicates and time stamps) in those behaviours.

We can accept this definition since it is the domain describer who decides which events to model and since it is anyway only a postulate: we are “observing the domain”!

**Definition: Event Designator::** The event  $\mathbb{E} : \{(\beta l_1, \sigma_1, P_1, \sigma'_1, \tau_1), (\beta l_2, \sigma_2, P_2, \sigma'_2, \tau_2), \dots, (\beta l_n, \sigma_n, P_n, \sigma'_n, \tau_n)\}$  consists of  $n$  event designators  $(\beta l_i, \sigma_i, P_i, \sigma'_i, \tau_i)$ , that is: an event designator is that kind of quintuplet.

**Example:** *Withdrawal of funds from an account* (i.e., a certain action) leads to either of two events: either *the remaining balance is above or equal to the credit limit*, or *it is not* ■

The withdrawal effects a state change (into state  $\sigma'$ ), but “below credit limit” event does not cause a further state change (that is:  $\sigma = \sigma'$ ). In the latter case that event may trigger a corrective action but the ensuing state change (from some (possibly later state)  $\sigma''$  to, say,  $\sigma'''$ , that is,  $\sigma'''$  is usually not a “next state” after  $\sigma'$ ).

**Example:** *A national (or federal) bank changes its interest rate.* This is an action by the behaviour of a national (or federal) bank, but is seen as an event by (the behaviour of) a(ny) local bank, and may cause such a bank to change (i.e., an action) its own interest rate ■

**Example:** *A local bank goes bankrupt at which time a lot of bank clients loose a lot of their money* ■

Some events are explicitly willed, and are “un-interesting”. Other events are “surprising”, that is, are not willed, and are thus “interesting”. Being interesting or not is a pragmatic decision by the domain describer.

**“First Class” Entities:** Before closing this section, Sect. 3.3, we shall “lift” events to be first class entities !

<sup>7</sup> The predicates can all “temporarily”, for purposes of “identity”, be set to **true**.

<sup>8</sup> The time stamps can all “temporarily”, for purposes of “identity”, be set to the smallest time interval within which all time stamps of the event are included.

### 3.4 Behaviours

A simple, sequential behaviour,  $\beta$ , is a possibly infinite, possibly empty sequence of actions and events.

**Example:** *The movement of a single vehicle between two time points forms a simple, sequential behaviour* ■

We shall later construct composite behaviours from simple behaviours. In essence such composite behaviours is “just” a set of simple behaviours. In such a composite behaviour one can then speak of “kinds” of consecutive or concurrent behaviours. Some concurrent behaviours can be analysed into communicating, joined, forked or “general” behaviours such that any one concurrent behaviour may exhibit two or more of these ‘kinds’.

Section 7.3 presents definitions of composite behaviours.

**“First Class” Entities:** Before closing this section, Sect. 3.4, we shall “lift” behaviours to be first class entities!

### 3.5 First-class Entities

Operations are considered designators of actions. That is, they are action descriptions. We do not, in this paper, consider forms of descriptions of events (labels) and behaviours. In that sense, of not considering, this section is not “completely” symmetrical in its treatment of operations, actions, events and behaviours as first-class entities. Be that as it may.

**Operations as Entities:** Operations may be (parametrised by being) applicable to operation entities — and we then say that the operations are higher-order operations: Sorting a set of rail units according to either length or altitude implies one sorting operation with either a select rail unit length or a select altitude parameter. (The ‘select’ is an operation.)

**Actions as Entities:** Similarly operations may be (parametrised by being) applicable to actions: Let an action be the invocation of the parametrised sorting function — cf. above. Our operation may be that of observing storage performance. There are two sorting functions: one according to rail unit length, another according to rail unit altitude. We may now be able, given the action parameter, to observe, for example the execution time!

**Events as Entities:** Operations may be (parametrised by being) applicable to a set of event entities: Recall that events are dynamic, instantaneous ‘quantities’. A ‘set of event entities’ as a parameter can be such a quantity. One could then inquire as to which one or more events occurred first or last, or, if they had a time duration, which took the longest to occur! This general purpose event handler may then be further parametrised by respective rail or air traffic entities!

**Behaviours as Entities:** Finally operations may be (parametrised by being) applicable to behaviours. We may wish to monitor and/or control train traffic. So the monitoring & control operation is to be real-time parametrised by train traffics. Similar for air traffic, automobile performance, etc.

• • •

We are not saying that a programming language must provide for the above structures. We are saying that, in a domain, as it is, we can “speak of” these parametrisations. Therefore we conclude that actions, events and behaviours — that these dynamic entities which occur in “real-time” — are entities. Whether we can formalise this “speaking of” is another matter.

### 3.6 The Ontology: Entities and Properties

On the background of the above we can now summarise our ontology: it consists of (“first class”) entities inseparable from their properties. We hinted at properties, in a concrete sense above: as something to which we can ascribe a name, a type and a value. In contrast to common practice in treatises on ontology [27, 33, 48, 52, 67], we “fix” our property system at a concrete modelling level around the value types of atomic simple entities (numbers, Booleans, characters, etc.) and composite simple entities (sets, Cartesians, lists, maps and functions); and at an abstract, orthogonal descriptive level, following Jackson [44], static and inert, active (autonomous, biddable, programmable) and reactive dynamic types; continuous, discrete and chaotic types; tangible and intangible types; one-, two-, etc.,  $n$ -dimensional types; etc.

Ontologically we could claim that an entity exists qua its properties; and the only entities *that we are interested in* are those that can be formalised around such properties as have been mentioned above.

## 4 Simple Atomic and Composite Entities

Entities are either atomic or composite. The decision as to which entities are considered what is a decision taken solely by the describer. The decision is based on the choice of abstraction level being made.

### 4.1 Simple Attributes — Types and Values

With any entity whether atomic or composite, and then also with its sub-entities, etcetera, one can associate one or more simple attributes.

- *By a **simple attribute** we understand a pair of a designated **type** and a named **value**.*

Attributes are not entities: they merely reflect some of the properties of an entity. Usually we associate a name with an entity. Such an association is purely a pragmatic matter, that is, not a syntactic and not a semantic issue.

## 4.2 Atomic Entities

- By an **atomic entity** we intuitively understand a simple attributes entity which ‘cannot be taken apart’ (into other, the sub-entities).

**Example:** We illustrate attributes of an atomic entity.

| Atomic Entity: <b>Bus Ticket</b> |                                |
|----------------------------------|--------------------------------|
| Type                             | Value                          |
| Bus Line                         | Greyhound                      |
| From, Departure Time             | San Francisco, Calif.: 1:30 pm |
| To, Arrival Time                 | Reno, Nevada: 6:40 pm          |
| Price                            | US \$ 52.00                    |

‘Removing’ attributes from an entity destroys its ‘entity-hood’, that is, attributes are an essential part of an entity.

## 4.3 Composite Entities

- By a **composite entity** we intuitively understand an entity (i) which “can be taken apart” into sub-entities, (ii) where the composition of these is described by its **mereology**, and (iii) which further possess one or more attributes.

**Example:** We “diagram” the relations between sub-entities, mereology and attributes of transport nets.

| Composite Entity: <b>Transport Net</b>   |   |
|--|---|
| <b>Sub-entities:</b> Links<br>Hubs   |   |
| <b>Mereology:</b> “set” of one or more $\ell$ (inks) and<br>“set” of two or more $h$ (hub’s)<br>such that each $\ell$ (ink) is delimited by two $h$ (hub’s)<br>and such that each $h$ (hub) connects one or more $\ell$ (inks) |   |
| <b>Attributes</b>  |   |
| <b>Types:</b>  | <b>Values:</b>  |
| Multimodal<br>Transport Net of<br>Year Surveyed  | <i>Rail, Roads, Sea_Lane, Air_Corridor</i><br><i>Denmark</i><br><i>2008</i> |

## 4.4 Discussion

**Attributes:** Domain entity attributes whether of atomic entities or of composite entities are modelled as a set of pairs of distinctly named types and values. It may be that such entity attributes, some or all, could be modelled differently, for example as a map from type names to values, or as a list of pairs of distinctly named types and values, or as a Cartesian of values, where

the position determines the type name — somehow known “elsewhere” in the formalisation, etcetera

But it really makes no difference as remarked earlier: one cannot really remove any one of these attributes from an entity.

**Compositions:** We formally model composite entities in terms of its immediate sub-entities, and we model these as observable, usually as sets, immediately from the entity (cf. `obs_Hs`, `obs_Ls`, `N`). In the example composite entity (nets) above the net can be considered a graph, and graphs,  $g:G$ , are, in Graph Theory typically modelled, for example, as

```

type
  V
  G = (V × V)-set

```

where vertexes ( $v:V$ ) are thought of a names or references.

We shall comment on such a standard graph-theoretic model in relation to a domain model which somehow expresses a graph: First it has abstracted away all there may otherwise be to say about what the graph actually is an abstraction of. In such models we model edges in terms of pairs of vertexes. That is: edges do not have separate “existence” — as have segments. In other words, since we can phenomenologically point to any junction and a segment we must model them separately, and then we must describe the mereology of nets separate from the description of the parts.

## 5 Atomic and Composite Operations

Entities are either atomic or composite. The decision as to which operations are considered what is a decision sôlely taken by the describer.

### 5.1 Signatures — Names and Types

With any operation whether atomic or composite, and then also with its sub-operations, etcetera, one can associate a signature which we represent as a triple: the name of the operation, the arguments to which the operation is applicable, and the result, whether atomic or composite.

- *By an **argument** and a **result** we understand the same as an attribute or an entity.*

### 5.2 Atomic Operations

We understand operations as functions in the sense of recursive function theory [51]<sup>9</sup> but extended with postulated primitive observer (`obs_...`), constructor

<sup>9</sup> See: <http://www-formal.stanford.edu/jmc/basis1/basis1.html>

(mk...) and selector (s...) functions, non-determinacy<sup>10</sup> and non-termination (i.e., the result of non-termination is a well-defined **chaotic** value).

- *By an **atomic operation** we intuitively understand an operation which ‘cannot be expressed in terms of other (phenomenological or conceptual), primitive recursive functions.*

**Example Atomic Operations:** The operation of obtaining the length of a segment, `obs_Lgth`, is an atomic operation. The operation of calculating the sum, `sum`, of two segment lengths is an atomic operation.

```

type
  Lgth
value
  obs_Lgth: L → Lgth,
  sum: Lgth × Lgth → Lgth

```

### 5.3 Composite Operations

- *By a **composite operation** we intuitively understand an operation which can best be expressed in terms of other (phenomenological or conceptual) primitive recursive functions, whether atomic or themselves composite.*

**Example Composite Operations:** Finding the length of a route, `R_Lgth`, where a route is a sequence of segments joined together at junctions is a composite operation — its sub-operations are the operation of observing a segment length from a segments, `obs_length`, and the recursive invocation of `route_length`. Finding the total length of all segments of a net is likewise a composite operation.

```

value
  length: L* → Lgth,
  zero_lgth:Lgth,
  length(⟨⟩) ≡ zero_lgth,
  length(ℓ^ℓ') ≡ sum(ℓ,ℓ')

```

**The Composition Homomorphism:** Usually composite operations are applied to composite entities. In general, we often find that the functions applied to composite entities satisfy the following homomorphism:

$$\mathcal{G}(e_1, e_2, \dots, e_m) = \mathcal{H}(\mathcal{G}(e_1), \mathcal{G}(e_2), \dots, \mathcal{G}(e_n))$$

where  $\mathcal{G}$  and  $\mathcal{H}$  are suitable functions.

• • •

---

<sup>10</sup> Hinted at in [51] as ambiguous functions, cf. Footnote 9 on the preceding page.

**Example:** Consider the *Factorial* and the *List Reversal* functions. This example is inspired by [50]. Let  $\phi$  be the sentence:

$$\exists F \bullet ((F(a) = b) \wedge \forall x \bullet (p(x) \supset (F(x) = H(x, F(f(x))))))$$

which reads: there exists a mathematical function  $F$  such that,  $\bullet$ , the following holds, namely:  $F(a) = b$  (where  $a$  and  $b$  are not known), and,  $\wedge$ , for every (i.e., all)  $x$ , it is the case,  $\bullet$ , that if  $p(x)$  is true, then  $F(x) = H(x, F(f(x)))$  is true.

There are (at least) two possible (model-theoretic) interpretations of  $\phi$ . In the first interpretation, we first establish the type  $\Omega$  of natural numbers and operations on these, and then the specific context  $\rho$ :

$$\begin{aligned} & [F \mapsto \text{fact}, a \mapsto 1, b \mapsto 1, f \mapsto \lambda \text{ n.n} - 1, \\ & \quad H \mapsto \lambda \text{ m}.\lambda \text{ n.m} + \text{n}, \\ & \quad p \mapsto \lambda \text{ m.m} > 0 ] \end{aligned}$$

We find that  $\phi$  is true for the factorial function, **fact**. In other words,  $\phi$  characterises properties of that function.

In the second interpretation we first establish the type  $\Omega$  of lists and operations on these: and then the specific context  $\rho$ :

$$\begin{aligned} & [F \mapsto \text{rev}, a \mapsto \langle \rangle, b \mapsto \langle \rangle, f \mapsto \mathbf{tl}, \\ & \quad H \mapsto \lambda \ell_1.\lambda \ell_2.\ell_1 \widehat{\mathbf{hd}} \ell_2, \\ & \quad p \mapsto \lambda \ell.\ell \neq \langle \rangle ] \end{aligned}$$

And we find that  $\phi$  is true for the list reversal function, **rev**, as well. In other words,  $\phi$  characterises properties of that function, and the two  $H$ s express a mereological nature of composition ■

## 6 Atomic and Composite Events

Usually events are considered atomic. But for the sake of argument — that is, as a question of scientific inquiry, of the kind: *why not investigate*, seeking “orthogonality” throughout, now that it makes sense to consider atomic and composite entities and operations — we shall explore the possibility of considering composite events.

Let us first recall that we model an event by:  $\mathbb{E} : \{(\beta\ell_1, \sigma_1, P_1, \sigma'_1, \tau_1), (\beta\ell_2, \sigma_2, P_2, \sigma'_2, \tau_2), \dots, (\beta\ell_n, \sigma_n, P_n, \sigma'_n, \tau_n)\}$ , where  $\mathbb{E}$  is just a convenient name for us to refer to the event,  $\beta\ell_i$  is the label of a behaviour  $\beta_i$ ,  $\sigma_i$  and  $\sigma'_i$  are (before event, resp. after event) states (of behaviour  $\beta_i$ ),  $P_i$  is a predicate which characterises the event as seen by behaviour  $\beta_i$ , and  $\tau_i$  is a time,  $t_i$ , or a time interval,  $[t_{i_b}, t_{i_e}]$ , time stamp.

## 6.1 Atomic Events

**Examples:** (i)  $\mathbb{E}_1$ : a vehicle “drives off” a net link at high velocity; (ii)  $\mathbb{E}_2$ : a link “breaks down”: (ii.a)  $\mathbb{E}_{2_1}$ : a bridge collapses, or (ii.b)  $\mathbb{E}_{2_2}$ : a mud slide covers the link ■ That is  $\mathbb{E}_2$  is due to either  $\mathbb{E}_{2_1}$  or  $\mathbb{E}_{2_2}$ .

One can discuss whether these examples really can be considered atomic: (ii.a) the bridge may have collapsed due to excess load and thus the moment at which the load exceeded the strength limit could be considered an event causing the bridge collapse; (ii.b) the mud slide may have been caused by excessive rain due to rainstorm gutters exceeding their capacity and thus the moment at which capacity was exceeded could be considered an event causing the mud slide.

We take the view that it is the decision of the domain describer to “fix” the abstraction level and thus decide whether the above are atomic or composite events.

In general we could view an event, such as summarised above, which involves two or more distinct behaviours as a composite event. We shall take that view.

## 6.2 Definitions: Atomic and Composite Events

**Definition: Atomic Event:** An *atomic event* is either a *single [atomic] internal event*:  $\{(\beta l_i, \sigma_i, P_i, \sigma'_i, \tau_i)\}$ , that is, consists of just one event designator, or is a *single [atomic] external event*, that is, is a pair event designators where one of these involves the external behaviour:  $\{(\beta \chi^l, \sigma_{\text{nil}}, \mathbf{true}, \sigma_{\text{nil}}, \tau_\chi), (\beta l_i, \sigma_i, P_i, \sigma'_i, \tau_i)\}$ , that is, consists of two event designators, an external and an internal ■

**Definition: Composite Event:** A *composite event* is an event which consists of two or more internal “identical” event designators, that is, event designators from two or more simple, non-external behaviours, and possibly also an event designator from an external behaviour “identical” to these internal event designators ■

## 6.3 Composite Events

**Examples:** (i) *two or more cars crash* and (ii) *a bridge collapse causes one or more cars or bicyclists and people to plunge into the abyss* ■

**Synchronising Events:** Events in two or more simple behaviours are said to be synchronising iff they are identical.

**Example:** *Two cars crashing means that the surfaces of the crash is a channel on which they are synchronising and that the messages being exchanged are “you have crashed with me”* ■



**Sub-Events:** A composite event defines one or more sub-events.

**Definition Sub-event:** An event  $\mathbb{E}_s:eds'$ , is a sub-event of another event  $\mathbb{E}:eds$ , iff  $eds' \subset eds$ , that is the set  $eds'$  of event designators of  $\mathbb{E}_s$  is a proper subset  $eds$  of the event designators of  $\mathbb{E}$  ■

**Sequential Events:** One way in which a composite event is structured can be as a “sequence” of “follow-on” sub-events. One sub-event:  $\mathbb{E}_{s_{12}}: \{(\beta\ell_1, \sigma_1, P_1, \sigma'_1, \tau_1), (\beta\ell_2, \sigma_2, P_2, \sigma'_2, \tau_2)\}$ , for example, “leads on” to another sub-event:  $\mathbb{E}_{s_{23}}: \{(\beta\ell_2, \sigma''_2, P'_2, \sigma'''_2, \tau'_2), (\beta\ell_3, \sigma_3, P_3, \sigma'_3, \tau_3)\}$ , etcetera, “leads on” to a final event:  $\mathbb{E}_{s_{mn}}: \{(\beta\ell_m, \sigma''_m, P_m, \sigma'''_m, \tau_m), (\beta\ell_n, \sigma_n, P_n, \sigma'_n, \tau_n)\}$ . The “leads on” relation should appear obvious from the above expressions.

**Example:** *The multiple-car crash in which the cars crash, “immediately” one after the other, as in a accordion movement* ■ (This is, of course, an idealised assumption.)

**Embedded Events:** Another way in which a composite event is structured is as an “iteratively” (or finite “recursively”) embedded “repetition” of (albeit distinct) sub-events. Here we assume that the  $\tau s$  stand for time intervals and that  $\tau s' \sqsubseteq \tau s$  it means that the time interval  $\tau s'$  is embedded with  $\tau s$ , that is, let  $\tau s = [t_b, t_e]$  and  $\tau s' = [t'_b, t'_e]$ , then for  $\tau s' \sqsubseteq \tau s$  means that  $t_b \leq t'_b$  and  $t'_e \leq t_e$ . Now we postulate that one event (or sub-event)  $\mathbb{E}_i$  embeds a sub-event  $\mathbb{E}_{i_j}, \dots$ , embeds an “innermost” sub-event  $\mathbb{E}_{i_j \dots k}$ .

**Example:** The following represents an idealised description of how a computing system interrupt is handled.

- (i) A laptop user hits the ENTER keyboard key at time  $t_b$ .
- (ii) The computing system interrupt handler reacts at time  $t'_b$  ( $t_b \leq t'_b$ ), to the hitting of the ENTER keyboard key.
- (iii) The interrupt handler forwards, at time  $t''_b$ , the hitting of the ENTER keyboard key to the appropriate input/output handler of the computing system keyboard handler.
- (iv) The keyboard handler forwards, at time  $t'''_b$ , the hitting of the ENTER keyboard key to the appropriate application program routine.
- (v) The application program routine calculates an appropriate reaction between times  $t'''_b$  and  $t'''_e$ .
- (vi) The application program routine returns its reaction to the keyboard handler at time  $t''_e$ .
- (vii) The keyboard handler returns, at time  $t''_e$ , that reaction to the interrupt handler.
- (viii) The interrupt handler marks the interrupt as having been fully served at time  $t'_e$ ,
- (ix) while whatever (if anything that has been routed to, for example, the display associated with the keyboard) is displayed at time  $t_e$  ■

The pairs (i,ix), (ii,viii), (iii,vii) and (iv,vi) form pairwise embedded events: (ii,vii) is directly embedded,  $\sqsubseteq$ , in (i,ix), (iii,vii) is directly embedded,  $\sqsubseteq$ , in (ii,viii) and (iv,vi) is directly embedded,  $\sqsubseteq$ , in (iii,vii).

We have abstracted the time intervals to be negligible.

**Event Clusters:** A final way of having composite events, is for them, as a structure, to be considered a set of sub-events, each eventually involving a time or a time period that is “tightly” related to those of the other sub-events in the set and where the relation is not that of “follow-on” or embeddedness.

**Example:** A (i) car crash results in a (ii) person being injured, while a (iii) robber exploits the confusion to steal a purse, etcetera ■

## 7 Atomic and Composite Behaviours

Our treatment of behaviours in Sect. 3.4 was very brief. In this section it will be more detailed. First a preliminary.

### 7.1 Modelling Actions and Events

In modelling behaviours, we model actions by a triple,  $(\beta\ell, \alpha, \tau s)$ , consisting of a behaviour label,  $\beta\ell:\text{BehLbl}$ , an operation denotation,  $\alpha:[\dots \times]\Sigma \rightarrow \Sigma[\times\dots]$ , and a time stamp,  $\tau s$ . Events are modelled by as above.

### 7.2 Atomic Behaviours

Time-stamped actions and atomic events are the only atomic behaviours. We shall model atomic behaviours as singleton sequences of a time-stamped action or an event.

### 7.3 Composite Behaviours

**Simple Traces:** A simple (finite/infinite) trace,  $\tau$ , is a (finite/infinite) sequence of one or more time-stamped atomic actions and time-stamped (atomic or composite) events. Trace time stamps occur in monotonically increasing dense order, i.e., separated by consecutive (overall) time stamps. That is, two traces may operate not only on different clocks, but have varying time intervals between consecutive actions or events. The “overall” time stamp of a composite event is the smallest time interval which encompasses all time and time stamps of event designators of the composite event.

**Simple Behaviours:** A simple behaviour,  $\beta$ , is a simple trace of length two or more.

**Example:** The movement of two or more vehicles between two time points forms a simple, concurrent behaviour ■

One can usually decompose a simple behaviour into two or more consecutive behaviours, and hence one can compose a consecutive behaviour from two or more simple behaviours. Consecutive behaviours are simple behaviours.

*Consecutive Behaviours:* A consecutive behaviour is a pair of simple behaviours, of which the first is finite, such that the time stamp of the first action or event of the second behaviour is consecutive to the time stamp of the last action or event of the first behaviour, cf. Fig. 1 on the next page.

**Example:** *A train travel, seen from the point of view of one train passenger, from one city to another, involving one or more train changes, and including the train passenger’s behaviours at train stations of origin, intermediate stations and station of destination as well as during the train rides proper, forms a consecutive behaviour ■*

*Concurrent Behaviours:* A concurrent behaviour is a set of two or more simple behaviours  $\{\beta_1, \beta_2, \dots, \beta_n\}$  such that for each behaviour  $\beta_i \in \{\beta_1, \beta_2, \dots, \beta_n\}$  there is a set of one or more different behaviours  $\{\beta_{i_j}, \beta_{i_k}, \dots, \beta_{i_\ell}\} \subseteq \{\beta_1, \beta_2, \dots, \beta_n\}$  such that there is a set of one or more consecutive (dense) time stamps that are shared between behaviours  $\beta_i$  and  $\{\beta_{i_j}, \beta_{i_k}, \dots, \beta_{i_\ell}\}$ .

**Example:** *The movement of two vehicles between two time points (i.e., in some interval) forms a concurrent behaviour ■*

Concurrent behaviours come in several forms. These are defined next.

*Communicating Behaviours:* A communicating behaviour is a concurrent behaviour in which two or more (simple) behaviours contain identical (modulo predicate and time stamp) events.

**Example:** *The movement of two vehicles between two time points (i.e., in some interval), such that, for example, the two vehicles, after some time point in the interval, at which both vehicles have observed their “near-crash”, keeps moving along, may be said to be a simple, cooperating behaviour. Their “near-crash” is an event. In fact the vehicles may be engaged in several such “near-crashes” (drunken driving!) ■*

**Example:** *The action of a vehicle, at a hub, which effects both a turning to the right down another link, and a sequence of one or more gear changes, throttling down, then up, the velocity, while moving along in the traffic, forms a general, structured behaviour ■*

**Example:** *A crash between two vehicles defines an event with the two vehicles being said to be synchronised and exchanging messages at that event ■*

*Joined Behaviours:* A joined behaviour is a pair of a finite set,  $\{\beta_1, \beta_2, \dots, \beta_n\}$ , of finite (“first”) simple behaviours and a (“second”) simple behaviour, such that the time stamp of the first action or event of the second behaviour is consecutive to the time stamp of the last action or event of each of the the first behaviours. You can think of the joined behaviour as pictured in Fig. 1 on the following page.

**Example:** This example assumes a mode of travel by vehicles in which they (sometimes) travel in platoons, or convoys, as do military vehicles and — maybe future private cars. *A behaviour which starts with  $n$  ( $n$  being two or more) vehicles travelling by themselves, as  $n$  concurrent behaviours; where*

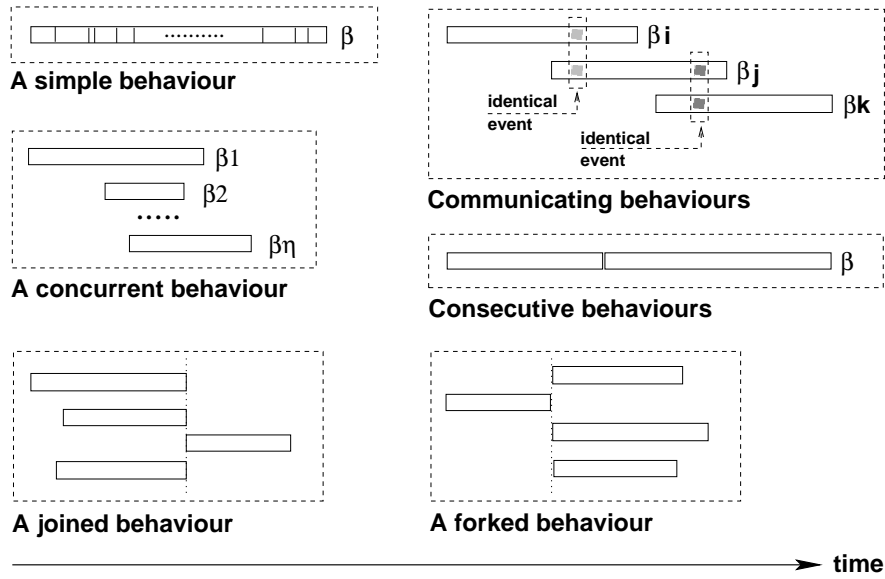


Fig. 1. Two simple and four composite behaviours  
 Each rectangle designates a simple behaviour. Figure indicates 17 such

*independent vehicles, at one time or another, join into convoy behaviours involving two or more vehicles, form a joined behaviour ■*

**Forked Behaviours:** A forked behaviour is a pair of a finite (“first”) simple behaviour  $\beta$  and a finite set,  $\{\beta_1, \beta_2, \dots, \beta_n\}$ , of (“second”) simple behaviours, such that the time stamp of the first action or event of each of the second behaviours is consecutive to the time stamp of the last action or event of the first behaviour. You can think of the joined behaviour as pictured in Fig. 1.

**Example:** Continuing the example just above: *A behaviour which starts as the joined, convoy behaviour of two or more (i.e.,  $n$ ) vehicles which then proceeds by individual vehicles, at one time or another, leaving the convoy, i.e., “forking out” into concurrent behaviours, forms a forked behaviour ■*

#### 7.4 General Behaviours

We claim that any set of behaviours can be formed from atomic behaviours by applying one or more of the compositions outlined above: simple, concurrent, communicating, consecutive, joined and forked behaviours.

By “any set of behaviours” you may well think of any multi-set of time stamped actions and time stamped events, i.e., of atomic behaviours. From this set one can then “glue” together one or more behaviours first forming a set of simple behaviours; then concurrent behaviours; then identifying pos-

sible communicating behaviours; then possibly joining and forking suitable behaviours, etc.

There may very well be many “solutions” to such a “gluing” construction from a basic set of atomic behaviours.

## 7.5 A Model of Behaviours

**type**

```
ActBeh, EvnBeh,
Beh = ABeh|CBeh,
ABeh = ActBeh|EvnBeh,
CBeh = fSimBeh|ifSimBeh|CurBeh|ComBeh|CnsBeh|FrkBeh|JoiBeh,
fSimBeh == mkSi(s_sb:ABeh*),
ifSimBeh == mkSi(s_sb:ABehω),
CurBeh == mkConc(s_cb:SimBeh-set),
ComBeh == mkComm(s_cb:SimBeh-set),
CnsBeh == mkCons(s_fst:fSimBeh,s_lst:ifSimBeh),
FrkBeh == mkFork(s_fst:fSimBeh,s_lst:ifSimBeh-set),
JoiBeh == mkJoin(s_fst:fSimBeh-set,s_lst:ifSimBeh)
```

**value**

```
wf_Beh: Beh → Bool
wf_Beh(beh) ≡ ...
```

## 8 Mereology and Compositionality Concluded

### 8.1 The Mereology Axioms

We wish to explain the compositionality constructs of simple entities (Sect. 8.2), operations (Sect. 8.3), events (Sect. 8.4) and behaviours (Sect. 8.5), where the references are to sections where the compositionality constructs are informally summarised. We wish that the explanation be in terms of the predicates of known axiomatisations of mereology, that is, of proposed such mereologies. We refer to Appendices A.1 on page 42 and A.2 on page 44 where such predicates are brought forward. Let  $x$ ,  $y$ , and  $z$  denote “first class” entities. Then:

1.  $\mathcal{P}xy$  expresses that  $x$  is a part of  $y$ ;
2.  $\mathcal{PP}xy$  expresses that  $x$  is a proper part of  $y$ ;
3.  $\mathcal{O}xy$  expresses that  $x$  and  $y$  overlap;
4.  $\mathcal{U}xy$  expresses that  $x$  and  $y$  underlap;
5.  $\mathcal{C}xy$  expresses that  $x$  is connected to  $y$ ;
6.  $\mathcal{DC}xy$  expresses that  $x$  is disconnected from  $y$ ;
7.  $\mathcal{DR}xy$  expresses that  $x$  is discrete from  $y$ ;
8.  $\mathcal{TP}xy$  expresses that  $x$  is a tangential part of  $y$ ; and
9.  $\mathcal{NTP}xy$  expresses that  $x$  is a non-tangential part of  $y$ .

## 8.2 Composite Simple Entities

**Mereology:** The part-whole mereological relations of composite simple entities are typically expressed by such defining phrases as: (i) “An  $x$  consists of a set of  $ys$ ” (modelled by  $X=Y\text{-set}$ ); (ii) “an  $x$  consists of a grouping of a  $y$ , a  $z$ , ... and a  $u$ ” (modelled by  $X=Y \times Z \times \dots \times U$ ); (iii) “an  $x$  consists of a list of  $ys$ ” (modelled by  $X=Y^*$ ); (iv) “an  $x$  consists of an association of  $ys$  to  $zs$ ” (modelled by  $X=Y \xrightarrow{m} Z$ ); and some more involved phrases, including recursively expressed ones.

Usually such defining phrases define too much. In such cases further sentences are needed in order to properly delimit the class of  $xs$  being defined.

**Example:** 14. A bus time table lists the bus line name 15. and one or more named journey descriptions, that is, journey names are associated with (maps into) journey descriptions. 16. Bus line and journey names are further undefined. 17. A journey description sequence of two or more bus stop visits. 18. A bus stop visit is a triple: the name of the bus stop, the arrival time to the bus stop, and the departure time from the bus stop. 19. Bus stop names are hub identifiers. 20. A bus time table further contains a description of the transport net. 21. The description of the transport net of the transport net, associates (that is, maps) each bus stop name hub identifier to a set of one or more bus stop name hub identifiers. 22. A bus time table is well-formed iff 23. adjacent bus stop visits name hubs that are associated in the transport net description; 24. arrival times are before departure times; etc.

### type

- 16.  $BLNm, JNm$
- 14.,20.  $BTT' = BLNm \times NmdBusJs \times NetDescr$
- 22.  $BTT = \{ | btt:BTT' \cdot wf\_BTT(btt) | \}$
- 15.  $NmdBusJs = JNm \xrightarrow{m} BusJ$
- 17.  $BusJ = BusStopVis^*$
- 18.  $BusStopVis = Time \times HI \times Time$
- 21.  $NetDescr = HI \xrightarrow{m} HI\text{-set}$

### value

- 22.  $wf\_BTT: BTT \times NetDescr \rightarrow \mathbf{Bool}$
- $wf\_BTT(\_,jrns,nd) \equiv$
- $\quad \forall bj:BusJ \cdot bj \in \mathbf{rng} \ jrns \Rightarrow$
- $\quad \quad \forall (at,hi,dt):BusStopVis \cdot (at,hi,dt) \in \mathbf{elems} \ bj \Rightarrow$
- $\quad \quad \quad hi \in \mathbf{dom} \ nd \wedge at < dt \wedge \dots$

The well-formedness predicate expresses part of the mereology of how bus time tables are composed. Note that we have not said that net description is commensurate with the actual transportation net ■

That is, we go from regular via context free to context sensitive and even generally computable or, alas, not necessarily computable forms. Thus there are rich opportunities to study suitable subsets of natural language mereology descriptions.

For the specific example of transportation nets, and as formalised in Sect. 1, we can prove that the following axiom system predicates hold as theorems:

- $\mathcal{PP}xy$  (Item 2 on page 21) holds for  $x$ =links or hubs of net,  $n$ , and  $y=n$ ;
- $\mathcal{C}xy$  (Item 5 on page 21) holds for such links  $x$  which connect hubs  $y$ , respectively such hubs  $x$  from which links  $y$  emanate; and
- $\mathcal{TP}xy$  (Item 8 on page 21) holds for such links  $x$  which connect hubs  $y$ , respectively such hubs  $x$  from which links  $y$  emanate.
- Let us introduce a notion of link/hub connectors. Any link  $x$  which is incident upon a hub  $y$  is said to define a connector  $j : \mathcal{J}$  such that for

**type J**

**value**  $\text{obs\_J}: (L|H) \rightarrow \text{J-set}$

**axiom**

$$\forall l:L, h:H \bullet \text{card } \text{obs\_J}(l)=2 \wedge \text{obs\_J}(l) \cap \text{obs\_J}(h) \neq \{\} \Rightarrow \text{obs\_HIs}(l) \cap \text{obs\_HI}(h) \neq \{\}$$

Now let  $x$  be the connector of link  $y$  and hub  $z$ , then  $\mathcal{O}xy$  and  $\mathcal{O}xz$  (Item 3 on page 21) hold.

- Etcetera.

**Compositionality:** The conventional compositionality principle implied a syntax of composite expressions and spoke of the semantics of composite expressions. We extend this principle to cover other than utterings of natural or formal specification and programming languages. We extend the principle to cover any structures that we may wish to contemplate.

To get the reader “tuned” to that idea we first give three, perhaps slightly “surprising” examples, and then return to examples in line with the main, the transport, example of this paper.

**Example:** *The design of a bread-toaster denotes the infinite set of all bread-toasters that satisfy the design, or the infinite set of all the production processes that construct such bread toasters, etc.* ■

**Example:** *The request from the marketing department of the producer of the bread toaster to the design department suggesting a bread toaster that satisfies certain market requirements denote the set of all bread-toaster designs that satisfy these market requirements, etc.* ■

**Example:** *The request from executive management to the marketing department requesting that measures be taken too win market share denote, amongst others, the kind of requests alluded to in the previous example* ■

Now to the examples that fit into the main example of this paper.

**Examples:** (i) *A meaning of a link could be the set of all paths that vehicles can traverse the link*, where a link path could be modelled as a triple of link connected hub identifiers and the link identifier. (ii) *A meaning of a hub*

could be the set of all paths that vehicles can traverse the hub, where a hub path could be modelled as a triple of link identifiers and the connecting hub identifier; (iii) *A meaning of a net could be the set of all routes through the net*, where a route is a suitable sequence of either link paths or of hub paths ■

**Compositionality of Simple Composite Entities:** The meaning of atomic entities are expressed by simple (recursive) functions.

The meaning of composite entities, in order to follow the principle of compositionality, must be a function of the meanings of the immediate sub-entities. Here the possibilities are “ad-infinitum” ! Classically, in computing, the principle of compositionality was first applied to programming, then to specification languages. Typically the meaning of an atomic statement, as a syntactic simple entity, of an imperative programming language was that of a function from environments to state to state transformers, so the meaning of the composition of two or more statements was then the function composition of the meaning of each statement. The meaning of a logic program could be modelled as a set of resolutions: bindings of identifiers to terms. The meaning of a parallel, say CSP [39, Hoare], program can be denotationally, that is, according to the principle of compositionality, for example, given either one of three kinds of semantics: in terms of traces, in terms of failures, and in terms of divergences — as all explained in [60, Roscoe, Chapter 8].

• • •

Whether all reasonably expressed meanings of all conceivable composite simple entities can be expressed compositionally is not known, well, is not knowable.

### 8.3 Composite Operations

**Mereology:** It appears that  $\mathcal{H}$  (in  $\mathcal{G}(e_1, e_2, \dots, e_m) = \mathcal{H}(\mathcal{G}(e_1), \mathcal{G}(e_2), \dots, \mathcal{G}(e_n))$ ), and the way in which  $\mathcal{G}$  distributes over  $(e_1, e_2, \dots, e_m)$ , in the abstract expresses the mereology of function composition. In the concrete the mereological nature of a given composite operation is reflected in the way in which it is structured from primitive recursive functions and other composite operations.

In the sense of recursive function theory it does not seem to make any sense to apply any of the 9 operators (Page 21) of Sect. 8.1.

**Compositionality:** The compositionality of functions is here taken to be expressed by the function composers of extended recursive function theory. Extended recursive function theory defines (i) constant entities,  $f()$  (i.e., values as zero-ary functions); (ii) variables,  $v$  (as functions from an environment to an entity); (iii) sets  $\{e_1, e_2, \dots, e_n\}$ , Cartesians  $(e_1, e_2, \dots, e_n)$ , and lists  $\langle e_1, e_2, \dots, e_n \rangle$  of entities; (iv) a finite set of primitive functions,  $f_n$ , of arity  $n$  and type  $f_n: \text{Entity}^* \rightarrow \text{Entity}$  (v) a finite set of primitive predicates,  $p_n$ , of arity  $n$  and type  $p_n: \text{Entity}^* \rightarrow \text{Bool}$  (vi) function composition  $f(g(e_1, e_2, \dots, e_n))$ ,



(vii) conditionals  $(c_1 \rightarrow e_1, c_2 \rightarrow e_2, \dots, c_n \rightarrow e_n)$ , (viii)  $\dots$ , and (ix)  $\dots$ . Recursive function theory then tells us how to interpret these forms in some context  $\rho$  which binds variables to entities and function and predicate function symbols to their functions and predicates. The extended recursive function theory is a simple encoding from recursive function theory. So compositionality of operations is explained by recursive function theory.

#### 8.4 Composite Events

**Mereology:** Mereologically events can be (i) sequenced, “connected” in time; (ii) “recursively” embedded, with the time interval for an “outer”, embedding event embracing the time interval for an immediately embedded event, and so forth; or can be (iii) clustered.

For specific, formalised examples of the three kinds of events it may then be possible to prove the following: (i)  $\mathcal{PP}xy$  where  $x$  is an event of either a sequenced event, or of a recursively embedding, or of a clustered event  $y$  ( $x$  is embedded in the recursively embedding event  $y$ ); (ii)  $\mathcal{C}xy$  where  $x$  and  $y$  are two consecutive events of a sequenced event  $z$  (that is, (ii:A)  $\mathcal{PP}xz \wedge \mathcal{PP}xz$ , (ii:B)  $\mathcal{TP}xy$  and (ii:C)  $\mathcal{U}xy$  (of  $z$ )), (ii:D) while  $\mathcal{DC}uv$  where  $(\mathcal{PP}uz \wedge \mathcal{PP}vz) \wedge \sim \mathcal{C}uv$ ; (iii)  $\mathcal{O}xy$  may hold for  $x$  and  $y$  being part of a clustered event  $z$  (that is,  $\mathcal{PP}xz \wedge \mathcal{PP}xz$  and  $\mathcal{U}xy$  (of  $z$ )). Etcetera.

**Compositionality:** Please note that we are not giving meaning to syntactic designators whose meaning is that of events. We are confronted with the issue of giving meaning, in some sense, to events. Simple such meanings are concerned with concrete event analysis: did to events occur at the same time, or in a given time period, or one before the other. For such analyses we refer to [45, Lamport]. We can think of obtaining more elaborate meanings for sequenced and recursively embedded events, but first we would need to build up a rather elaborate set of definitions, find elaborate examples, etcetera, and that would blow the size of this paper way out of proportion. So the best is to say, and this also applies to clustered events, here is an interesting research topic: to come up with compositionality interpretations for composite events !

#### 8.5 Composite Behaviours

**Mereology:** Let  $\{\beta_1, \beta_2, \dots, \beta_n\}$  be, for each case below, the suitable set of (simple) behaviours. From the point of view of the mereology of the composition of behaviours, such as we have modelled behaviours, there are the following composition operators:

- (i) creating simple behaviours,  $\beta_s:(f\text{SimBeh}|\text{ifSimBeh})$ , from suitable atomic ones  $(a_1, a_2, \dots, a_m, e_1, e_2, \dots, e_n)$ ;
- (ii) creating concurrent behaviours,  $\beta_{con}:\text{CurBeh}$ , from suitable simple behaviours;

- (iii) creating communicating behaviours,  $\beta_{com}:\text{ComBeh}$ , from a set of suitable simple behaviours;
- (iv) creating consecutive behaviours,  $\beta_{seq}:\text{CnsBeh}$ , from a set of suitable simple behaviours;
- (v) creating joined behaviours  $\beta:\text{JoiBeh}$  from a set of suitable simple behaviours; and
- (vi) creating forked behaviours,  $\beta:\text{FrkBeh}$ , from a set of suitable simple behaviours.

Given a specific composite behaviour is may then be possible to prove

- (i) For all distinct atomic behaviours  $\beta'_{\alpha\epsilon}$  and  $\beta''_{\alpha\epsilon}$  and for all simple behaviours,  $\beta_s$ , for which  $\mathcal{P}\beta'_{\alpha\epsilon}\beta_s \wedge \mathcal{P}\beta''_{\alpha\epsilon}\beta_s$  holds to prove that  $\mathcal{P}\mathcal{P}\beta'_{\alpha\epsilon}\beta_s \wedge \mathcal{P}\mathcal{P}\beta''_{\alpha\epsilon}\beta_s$ ; holds;
- (ii) for a set,  $\beta_s$ , of suitable simple behaviours to prove that  $\mathcal{P}\mathcal{P}\beta_i\text{mkConcurBeh}(\beta_s)$  holds for any  $\beta_i$  in  $\beta_s$ ;
- etcetera.

**Compositionality:** Behaviours are the meaning of domain descriptions, or or requirements prescriptions or software programs that specify concurrency. So the meaning functions that we might apply to behaviours would then typically be those of analysing behaviours: comparing behaviours, say with respect to efficiency, or to access to shared resources, or (say human) interface response times. We leave it to the reader to continue this line of thought.

## 9 Galois Connections

In the following sections, we look at some rôles Galois connections may have in relation to composition of entities. Galois connections is a fundamental mathematical concept from order- and lattice theory. The reason for bringing it up here is that it involves set-based composition of elements as well as the composition of dual, order-decreasing functions. However, this is just the reason why we considered Galois connections in the first place. In fact we want to argue that it is beneficial to incorporate nontraditional modelling aspects in order to get more insight; both in the discipline of domain engineering, and in a current domain of discourse.

In the following, we shall (i) define the notion of Galois connections, (ii) outline some of the different uses of that concept, and (iii) consider the concept in context of modelling composite entities (following the ontology presented in this paper). The sections are driven by examples.

**Example: Toasters and Their Designs.** *Let  $(d:D)$  be a design of a toaster  $(t:T)$ . From the design we may be able to produce a collection of different toasters because the design does not specify everything, and due to the fact that we could produce the “same” kind of toaster over and over again. Let us look at a “time glimp” and let  $(ts:T\text{-set})$  denote the set of such toasters*

obeying the design<sup>11</sup>. If we impose that “sequentially” further designs are all for the same toaster, then the number of toasters decreases<sup>12</sup> because they all need to satisfy the new designs too. Between the set of designs and the set of toasters they denote, is a Galois connection ■

**Example: Designs and Market Analysis.** The designs are also the denotation of something. It could be the market analysis indicating the need for certain toaster products — or more generally, for certain new kinds of kitchen equipment. Between the market analysis and the designs also stands a Galois connection; hence there is also a Galois connection between the market analysis and the toasters. The Galois connection (being an order-decreasing pair of functions) ensures that we can only produce toasters which obeys the designs, and that we can only design toasters which satisfy the needs outlined in the market analysis ■

### 9.1 Definition

**Definition 1.** A Galois connection is a dual pair of mappings  $(\mathcal{F}, \mathcal{G})$  between two orderings  $(P, \sqsubseteq)$  and  $(Q, \sqsubseteq)$ . Most often  $P$  and  $Q$  are ordered sets based on set-inclusion ( $\subseteq$ ) and this is also the version we shall use in this paper. In order to avoid misinterpretation, we write  $\sqsubseteq$  and not  $\leq$  or  $\subseteq$  as seen in other treatments of the subject. The mappings must be monotonically decreasing<sup>13</sup>:

**type**

$P, Q$

**value**

$\mathcal{F}: P\text{-set} \rightarrow Q\text{-set}$

$\mathcal{G}: Q\text{-set} \rightarrow P\text{-set}$

**axiom**

$\forall ps_1, ps_2:P\text{-set}, qs_1, qs_2:Q\text{-set} \bullet$   
 $ps_1 \sqsubseteq ps_2 \Rightarrow \mathcal{F} ps_2 \sqsubseteq \mathcal{F} ps_1,$

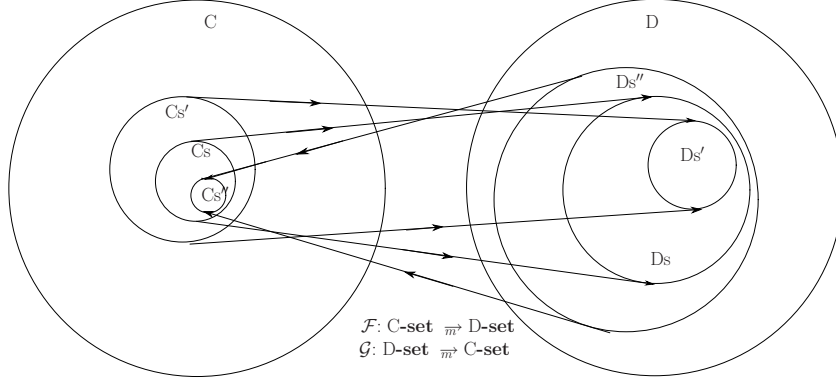
<sup>11</sup> We shall — as common in modelling — assume a *possible worlds semantics* in the sense that the collection of toasters are the toasters existing in one possible world. Are there more produced or some destroyed, it is another possible world. We shall not be further concerned with this, nor the many philosophical issues that can be claimed. We refer to [62] and [2] which among many other issues take up this discussion.

<sup>12</sup> Actually, the number could stay the same but that would mean including identical designs. In general, we shall not be that concerned with the equal-situation for that same reason.

<sup>13</sup> Note, that there are in fact two different definitions of Galois connections in the literature: the *monotone* Galois connection and the *antitone* Galois connection. We follow Ganter and Wille and assume the former [28].

$$\begin{aligned}
 qs_1 &\sqsubseteq qs_2 \Rightarrow \mathcal{G} qs_2 \sqsubseteq \mathcal{G} qs_1, \\
 ps_1 &\sqsubseteq \mathcal{G} \mathcal{F} ps_1, \\
 qs_1 &\sqsubseteq \mathcal{F} \mathcal{G} qs_1
 \end{aligned}$$

The dual ordering of Galois connections is illustrated on Figure 2.



**Fig. 2.** A Galois connection.

■

In [28, Ganter & Wille: FCA], the following Theorem is given on Galois connections:

**Theorem 1 (Galois Connection<sup>14</sup>).** For every binary relation  $R \subseteq M \times N$ , a Galois connection  $(\varphi_R, \psi_R)$  between M and N is defined by

$$\begin{aligned}
 \varphi_R X &:= X^R \quad (= y \in N \mid xRy \text{ for all } x \in X) \\
 \psi_R Y &:= Y^R \quad (= x \in M \mid xRy \text{ for all } y \in Y).
 \end{aligned}$$

From the above, we see that all  $y$  must stand in the relation  $R$  to each  $x$  in order for the connection to hold. However,  $R$  could mean “does not stand in a relation to”. That would still yield a Galois connection but the domain knowledge it expresses is different. Let  $X$  be a collection of coffee cups and let  $Y$  be a collection properties concerning form, colour, texture and material. We may define  $R$  to be “coffee cup  $x$  has property  $y$ ”. However, we could also define it as “coffee cup  $x$  does not have property  $y$ ”. In both cases we would have a Galois connection. However, the latter may be somewhat strange from a classification point of view.

The notion of Galois connections has served as foundation for a variety of applications like order theory, the theory of dual lattices, and — in computer science — semantics of programming languages and program analysis.

<sup>14</sup> In [28] this is named Theorem 2.

However, it has also been utilized in a number of conceptualization principles. These principles are not pure mathematical treatments, but utilize Galois connections in specific domains. We shall look at three such areas in the following.

## 9.2 Concept Formation in Formal Concept Analysis [FCA]

In the area of formal concept analysis [28, Ganter & Wille: FCA], the notion of Galois connections is used as foundation for the lattice-oriented theory used for concept formation. In FCA, concepts are defined from a collection of objects by looking at which objects have common properties. The approach includes algorithms for automatic concept formation, given a collection of objects or a collection of properties. The fact that we can choose either to form concepts from objects (the extension of the concepts) or the properties (the intensions of the concepts) shows the duality between objects and properties.

## 9.3 Classification of Railway Networks

In [42, 40, 41] Ingleby et al. use Galois connections in order to classify railway networks. The approach is similar to the approach of concept formation in FCA, but Ingleby understands the notion of properties in a broader sense: a property of a route may be the segments involved in the route. Here Ingleby understands routes and segments in a safety–security sense as his quest is to cluster routes and segments such that the complexity of safety proof over the railway network, is reduced. That is, the Galois connection is used for defining cluster segments (in FCA, corresponding to concepts) such that the number of free variables are reduced when proving safety properties of software/hardware for instance.

## 9.4 Relating Domain Concepts Intensionally

In [23, 24, Eir], we utilized the notion of Galois connections for relating domain concepts intensionally. The domain concepts related were concepts that were not bound under subsumption; i.e. they are not specialization/generalization pairs.

Consider the domain concepts: *Budgets* and *Project Plans*. From a budget we can observe the set of project plans that can be executed within the financial restrictions of the budget. From a project plan we can observe the set of budgets that designate the necessary figures for executing the project plan. Generalizing this gives two interpretation functions: one from a set of budgets to the set of project plans that are all executable within the restriction of each budget in the set; and another from a set of project plans to the set of budgets that all designate the necessary expenses for executing each project plan.

The pair of interpretation functions is a Galois connection. This approach is utilized in order to suggest a modelling approach for relating domain concepts and placing their models (i.e. their abstractions) in conceptual structures. For the two concepts mentioned above, the conceptual structure maintains the systematics of concretising information from budgeting to project planning.

### 9.5 Further Examples

We may easily produce other examples of domain concept pairs of which the objects relate in some way. Consider the following examples:

**Example: Bus Time Tables and Traffic.** Let  $btt$  be some bus time table ( $btt:(bln,busjs,nd)$ ). To  $btt$  there corresponds a set of bus traffics,  $sobustrfs$ , on the net. Express such bus traffic as  $(bustrf,n)$  where  $(bustrf,n) \in sobustrfs$  and where  $bustrf$  is the time-varying function from buses to their positions on the net, and  $nd$  is related to  $n$  in some way (one is a net description, the other is “the” (or that) net). We furthermore stipulate that each bus traffic  $(bustrf,n)$  “obeys” the timetable  $(bln,busjs,nd)$ . To a set of timetables,  $sobustts$ , over the same net there corresponds the union set of all those sets of bus traffics,  $usobustrfs$ , that “obey” all timetables in  $sobustts$  ■

We seek to understand the relationship between  $sobustts$  and  $usobustrfs$  in terms of the concept of Galois connections.

**Example: Traffic and Buses – The Dual Case.** We reverse the relation. We start with a bus traffic  $(bustrf,nd)$  and can, by arguments similar to above, postulate a set of bus timetables,  $sobustts$  (on the same net), such that each bus timetable properly records the arrival and departure times of buses at bus stops on that net. We can then “lift” this relation  $((bustrf,nd),sobustts)$  to a relation from sets of bus traffics to the union set of sets of bus timetables ■

We seek to understand the relationship between  $sobustrfs$  and  $usobustts$  in terms of the concept of Galois connections.

The two examples above each define what we in 9.4 called *interpretation functions*. They are interpretation functions in the sense that they — in the domain — “interpret” the time table entities as traffic entities; and vice versa.

### 9.6 Generalisation

The element that these example have in common is that the values of one concept characterize the values of the other concept — in some way.

This is similar to FCA where we have a Galois connection between values and their common properties. However, in this case the properties are extrinsic properties. The budget relates to a specific set of project plans because it possesses the property of standing in a certain relation to these other values. The property is *extrinsic* as the property is possessed assuming the existence of other values; as opposed to *intrinsic* properties.

In a sense it means that we break the traditional distinction between values and properties as assumed in FCA. Furthermore, we utilize the same principles as utilized in denotational semantics — namely that we can assign meaning to values (e.g., a budget) and the meaning to composition of values (e.g., a set of budgets). The meaning of the composition is here more than the meanings of the individual parts because the composition of budgets (the budget set inclusion) implies a more narrow restriction of the set of executable project plans. It is so because combining two budgets has influence on the meaning in the sense that the meaning is the composition of the corresponding project plans as well (satisfying the Galois functions of being “decreasing”). Mereologically, what is added when composing a whole is actually the axioms in the Galois connection.

However, we go a little further than denotational semantics of programming languages because we may consider any domain concept a subject for defining a Galois connection.

Another observation is that the notion of Galois connection is *domain neutral*. The Galois connection is a general mathematical framework and hence not what contributes to *why* two concepts relate intensionally.

### 9.7 Galois Connections and Ontology

In the ontology presented throughout this paper, we have exercised the importance of compositionality. I.e. we have defined compositionality for each of the four entity parametrisations made. In this sections, we shall look at how these can be understood in the general, domain and ontology neutral framework of Galois connections. When we say that Galois connections in this sense are ontologically neutral it is not entirely true. Many ontologies — especially in philosophy — concerns the existence of (say) mathematical entities; hence also heavily touching (perhaps disturbing) the foundation on which Galois connections are defined. However, this is not our quest here. When we consider Galois connections ontologically neutral it is in fact similar to saying that they are neutral to the ontology of entities that we have suggested. Whether entities include mathematical entities or these is “outside”, that is, is outside the scope of this paper. For further exploration, we refer to [62].

If simple entities, events, behaviour and operations are all entities, it should imply that we can make the same considerations involving such values in Galois connections. We shall try to do so in the following, and we intend in that context to outline the issues as we go along. The important thing is, however, not whether Galois connections can be established, but whether the Galois connection complies with the current intuition as the connection between objects and their common properties does.

The traditional use of Galois connections — as ‘exercised order theory’ and as used by Ganter and Wille focuses on the properties that objects have in *common*. However, we may turn this order upside-down such that we look

at the total set of properties. This will be a Galois connection as well but in the case of domains, it expresses a different aspect. In some situations, it is natural to consider the former — in other situations, we may prefer the latter. This depends on how we perceive the domain and — perhaps also — the purpose of our domain model; i.e. our perspective.

By the above we indicate that the study of Galois connections in the context of domain engineering could be interesting because it has to do with how we choose to perceive, abstract, model and formalize the domain. Hence, what we present in the following may open up for such research areas for further clarification. We are, however, not saying that these *will* be interesting or that it does make sense to make distinctions like the one above. We just say that this area deserves further exploration.

Let us in the following assume that Galois connections concern relations between two ordered sets of entities and the essence that the entities in these sets characterize each other. The issue is now that in some cases, characterization usually obeys the axioms of Galois; but in some situations it may not.

### Composite Entities

We have already seen a couple of examples of Galois connections between two ordered sets of elements, where the ordering has been set-inclusion.

We assume the understanding of composite entities as presented in Sects. 4 and 8.2.

**Example: Hospital Staff and Rostering (I).** *Doctors and nurses forming surgery teams. From a team (possibly empty or singleton), we can observe the collection of time slots where they are all available. If we include more doctors and nurses, we will have a smaller set of time slots. And vice versa. This is an important domain aspect when we are going to talk about planning and staffing (either in domain descriptions and specifications, or in software requirements). This is a Galois connection ■*

**Example: Hospital Staff and Rostering (II).** *Again consider doctors and nurses forming surgery teams. From a team, we can observe the collection of possible surgeries they may perform. If we include more doctors and nurses, we increase the collection of surgeries. And vice versa ■*

This is not a Galois connection, though interesting from a domain perspective anyway.

### Composite Operations

We assume the understanding of composite entities as presented in Sects. 5 and 8.3.

**Example: Building Constructions and Parts.** *Consider a set of building constructions: molding of foundations, mounting of bricks into walls, and establishing the roof, etc. Then consider the set of building parts involved in a*



construction. By building parts, we shall both understand the materials and elements consumed by constructions, and the results of other constructions. Thus, a building part can be a specific brick, a pre-cast concrete wall, the foundation, etc. That is, the building parts are those either created, mounted on, changed in some way, or demolished. For each construction, we can observe the building parts involved: consumed or produced. Building constructions can be composite in the sense that one construction constructs the foundation and another construction mounts the walls on the foundation. The former construction is a function from certain amounts of sand, stone, cement and water, to a foundation (here we shall exclude the tools needed). The latter construction is a function from a foundation, a collection of bricks, water, cement and insulation, to the product consisting of foundation and walls. For a construction — atomic or composite — we can observe the building parts involved in all constructions. If we include more constructions in a composite construction, the building parts involved in all constructions will decrease ■

The connection between composite constructions and building parts involved is a Galois connection.

The connection is interesting when modelling the planning and scheduling of construction works as a crucial element is that construction workers cannot always work on the same building parts at the same time.

**Example: Building Operations and Consumed Materials.** Now consider the approach where for each building operation we observe the materials needed. For a collection of operations, we can likewise observe the total quantity of materials needed. That is, the total amount of sand, stones, bricks; the total quantity of beams, doors, windows of each type and measure; etc. Including more building operation will increase the amount and quantity of materials needed; simply because we then build more. Then we have a situation where the more operations we include, the more products. That is, set-inclusion of the operations implies an increase of the observed materials and parts. The reason is that we here that each building operation contributes with a result. Instead of considering the common materials as characterizing the composite operation, we shall consider that the complete set of materials involved characterize the composite operation ■

In a sense this is more natural as we then include all the aspects of the compositionality. However, in the present case, we do not have Galois connection because including more operations in the composite, implies including more materials and results. Hence, dual ordering is increasing; not decreasing.

### Composite Events

We assume the understanding of composite entities as presented in Sects. 6 and 8.4.

**Example: Traffic Accidents and Responsible Persons.** Consider a traffic accident. This is an event and for the accident, we can observe the collection

*of persons involved and of these the persons bearing some kind of responsibility in the accident. Assume that we look at a collection of traffic accidents. Here, we can observe the persons involved in all accidents and for these the ones being responsible for the accidents. Including more traffic accidents will reduce the number of persons involved in all accidents; hence, also the number of persons being responsible in all accidents ■*

The connection between sets of traffic accident events and the set of persons being responsible, is a Galois connection.

The connection may be interesting when modelling the analysis of traffic accident patterns and statistics which may influence the definition of insurance premium.

**Example: Traffic Accidents and Persons Involved.** *Now, consider traffic accidents as events again. From a traffic accident, we can observe the insurance policies of the involved persons. Likewise, from a collection of traffic accidents (i.e. a composite event being a cluster of individual events), we can observe the collection of persons involved in at least one of the accidents; that is, the total collection of persons involved in one or more of the accidents. If we include more accidents, the collection of persons involved will increase ■*

This is not a Galois connection. Though the connection may be interesting when modelling correlation between accidents.

We should also be able to construct examples for composite events being sequential or embedded.

### Composite Behaviours

We assume the understanding of composite entities as presented in Sects. 78.5.

**Example: Meetings and Applicable Rooms.** *Consider a collection of persons engaged in a meeting. We shall consider having a meeting a behaviour. The meeting can be composite in the sense that we may join two or more meetings held in the same time interval and involving the same persons. In the present case we shall consider behaviour composition as communicating. E.g. we may join department meetings for several company departments if the topic of the meetings is common and should be shared. From a meeting, we can observe the rooms applicable. We shall assume that a room is only applicable if it can host the number of meeting participants, has the equipment necessary for the meeting, etc. If we include more meeting behaviours in a composite behaviour, the collection of rooms applicable will decrease ■*

This is a Galois connection. The connection is interesting when planning collaborative work among meeting participants.

**Example: Engineering Work and Skills.** *Consider a collection of engineers engaged in a project. We shall consider their work a behaviour which is concurrent — perhaps also communicating to an extent. From each engineering behaviour we can observe the engineering skills utilized and practiced. If*

*we include a collection of work behaviours as a composite behaviour, it implies that we include more engineers and thus also more engineering skills* ■

This is not a Galois connection; though interesting when modelling skills, skills management, project communication and interaction, staffing, etc.

## 9.8 Galois Connections Concluded

So what went wrong in the cases where we did not have a Galois connection? Or we could ask: what did we explore by looking at the domain through Galois eyes? The examples examined above clearly shows that there are two different kinds of connections between entity compositions; hence, orderings.

- The former yields a Galois connection. It does so because composite entities of the one ordering are *all* characterizing composite entities of the other. Thereby, we believe to have outlined how Galois connections and ordering theory in general plays an important rôle in compositionality of entities.
- The latter does not yield a Galois connection as it is an order-preserving connection. In the examples examined we have seen a general pattern composition of the one kind of entity, yields composition (actually just set-inclusion) of the other kind of entity.

Both kinds of connections show that even though the connections (Galois being order-reversing and the order-preserving) are ontologically and domain neutral, they do express interesting domain intrinsics when it comes to compositionality. We suggest that the rôle, use and axioms/theorems of such ordering connections are explored further within the context of domain engineering. Furthermore, we encourage exploring other such concepts and their ability of promoting domain engineering as a discipline.

## 10 Conclusion

### 10.1 Ontology

Ontology plays an important rôle in studies of epistemology and phenomenology. In the time-honoured tradition of philosophical discourse philosophers present proposals for one or another ontology, and discusses these while usually not settling definitively on any specific ontology; and many issues are deliberately left open.<sup>15</sup> In this paper we cannot afford this “luxury”. Our objective is to clarify notions of ontology in connection with the use of specific ways of informally and formally describing domains where the formal description language is fixed.

<sup>15</sup> Such as whether properties of entities are themselves entities, etc.

Many of the issues of domain modelling evolve close to issues of metaphysics. We find [48, Michael J. Loux] *Metaphysics, a contemporary introduction*, [33, Pierre Grenon and Barry Smith] *SNAP and SPAN: Towards Dynamic Spatial Ontology*, [63, Peter Simons] *Parts: A Study in Ontology*, and [52, D. H. Mellor and Alex Oliver] *Properties*, relevant for a deeper study of the meta-physical issues of the current essay.

## 10.2 Mereology

Mereology has been given a more concrete interpretation in this paper compared to the “standard” treatments in the (mostly philosophical) literature. It seems that Douglass T. Ross [61] was among the first computing scientists to see the relevance of Leśniewski’s ideas [49, 66]. Too late for a study we found [57, Chia-Yi Tony Pi]’s 287 page PhD (linguistics) thesis: *Mereology in Event Semantics*. Perhaps it is worth a study.

## 10.3 Research Issues

The paper has touched upon many novel issues. Some are reasonably well established, at least from a programming methodological point of view. Several issues could benefit from some deeper study. We mention three.

**Compositionality:** A precise study of how composite functions, events and behaviours can be understood according to the principle of compositionality.

**Mereology:** A more precise presentation of a mereology axiom system for the kind of simple entities, function entities, event entities and behaviour entities outlined in Sects. 4–7.

**Ontology:** A more precise comparison of the “computability”-motivated ontology of this paper as compared with for example the ontological systems mentioned in [48, Michael J. Loux], [33, Pierre Grenon and Barry Smith], [63, Peter Simons] and [27, Chris Fox].

**Galois Connections:** A further study, going beyond that of [23, 24, Asger Eir], of relations between compositionally and **Galois connections**. For that study one should probably start with [37, Hoare and He].

• • •

That we have not really studied the compositionality issue as listed above is a major drawback of this paper but we needed to clarify first the nature of “compositeness” of events, functions and behaviours before taking up the future study of their compositionality.

## 10.4 Acknowledgement

The first author is most grateful to his former PhD student, Dr. Asger Eir, for his willingness to co-author this paper.

## 11 Bibliographical Notes

[11, to appear] gives a concise overview of domain engineering; [12, to appear] gives a “complete” example of domain and requirements engineering; and [10, to appear] relates domain engineering, requirements engineering and software design to software management. [9] presents a number of domain engineering research challenges.

## References

1. Jean-Raymond Abrial. *The B Book: Assigning Programs to Meanings*. Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, England, 1996.
2. Mark Balaguer. *Platonism and Anti-Platonism in Mathematics*. Oxford University Press, 1998.
3. Dines Bjørner. Programming in the Meta-Language: A Tutorial. In Dines Bjørner and Cliff B. Jones, editors, *The Vienna Development Method: The Meta-Language*, [15], LNCS, pages 24–217. Springer-Verlag, 1978.
4. Dines Bjørner. Software Abstraction Principles: Tutorial Examples of an Operating System Command Language Specification and a PL/I-like On-Condition Language Definition. In Dines Bjørner and Cliff B. Jones, editors, *The Vienna Development Method: The Meta-Language*, [15], LNCS, pages 337–374. Springer-Verlag, 1978.
5. Dines Bjørner. The Vienna Development Method: Software Abstraction and Program Synthesis. In *Mathematical Studies of Information Processing*, volume 75 of LNCS. Springer-Verlag, 1979. Proceedings of Conference at Research Institute for Mathematical Sciences (RIMS), University of Kyoto, August 1978.
6. Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
7. 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.
8. Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
9. 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.
10. Dines Bjørner. Believable Software Management. *Encyclopedia of Software Engineering*, 1(1):1–32, 2008. (This is a new journal, published by Taylor & Francis, New York and London, edited by Philip Laplante).
11. 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, 2008. Springer. To appear.
12. Dines Bjørner. From Domains to Requirements. In *Montanari Festschrift*, volume 5065 of *Lecture Notes in Computer Science* (eds. Pierpaolo Degano, Rocco De Nicola and José Meseguer), pages 1–30, Heidelberg, May 2008. Springer.

13. Dines Bjørner. *Software Engineering, Vol. I: The Triptych Approach, Vol. II: A Model Development*. To be submitted to Springer for evaluation, expected published 2009. This book is the basis for guest lectures at Techn. Univ. of Graz, Politecnico di Milano, University of the Saarland (Germany), etc., 2008–2009.
14. Dines Bjørner and Martin C. Henson, editors. *Logics of Specification Languages — see [59, 18, 22, 55, 34, 29, 54, 25, 35]*. EATCS Monograph in Theoretical Computer Science. Springer, Heidelberg, Germany, 2008.
15. Dines Bjørner and Cliff B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *LNCS*. Springer-Verlag, 1978. This was the first monograph on *Meta-IV*. [3, 4, 5].
16. Dines Bjørner and Cliff B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall, 1982.
17. Dominique Cansell and Dominique Méry. Logical Foundations of the B Method. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [58, 21, 56, 31, 53, 36] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
18. Dominique Cansell and Dominique Méry. *Logics of Specification Languages*, chapter The event-B Modelling Method: Concepts and Case Studies, pages 47–152 in [14]. Springer, 2008.
19. R. Casati and A. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.
20. Bowman L. Clarke. A calculus of individuals based on “connection”. *Notre Dame J. Formal Logic*, 22(3):204–218, 1981.
21. Răzvan Diaconescu, Kokichi Futatsugi, and Kazuhiro Ogata. CafeOBJ: Logical Foundations and Methodology. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [58, 17, 56, 31, 53, 36] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
22. Răzvan Diaconescu. *Logics of Specification Languages*, chapter A Methodological Guide to the CafeOBJ Logic, pages 153–240 in [14]. Springer, 2008.
23. Asger Eir. *Construction Informatics — issues in engineering, computer science, and ontology*. PhD thesis, Dept. of Computer Science and Engineering, Institute of Informatics and Mathematical Modeling, Technical University of Denmark, Building 322, Richard Petersens Plads, DK–2800 Kgs.Lyngby, Denmark, February 2004.
24. Asger Eir. *Formal Methods and Hybrid Real-Time Systems*, chapter Relating Domain Concepts Intensionally by Ordering Connections, pages 188–216. Springer (LNCS Vol. 4700, Festschrift: Essays in Honour of Dines Bjørner and Zhou Chaochen on the Occasion of Their 70th Birthdays), 2007.
25. John S. Fitzgerald. *Logics of Specification Languages*, chapter The Typed Logic of Partial Functions and the Vienna Development Method, pages 453–487 in [14]. Springer, 2008.
26. John S. Fitzgerald and Peter Gorm Larsen. *Developing Software using VDM-SL*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 1RU, England, 1997.
27. Chris Fox. *The Ontology of Language: Properties, Individuals and Discourse*. . CSLI Publications, Center for the Study of Language and Information, Stanford University, California, USA, 2000.
28. Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer-Verlag, January 1999. ISBN: 3540627715, 300 pages, Amazon price: US\$ 44.95.

29. Chris George and Anne E. Haxthausen. *Logics of Specification Languages*, chapter The Logic of the RAISE Specification Language, pages 349–399 in [14]. Springer, 2008.
30. Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hempstead, England, 1992.
31. Chris W. George and Anne E. Haxthausen. The Logic of the RAISE Specification Language. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [58, 17, 21, 56, 53, 36] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
32. Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbak Pedersen. *The RAISE Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hempstead, England, 1995.
33. Pierre Grenon and Barry Smith. SNAP and SPAN: Towards Dynamic Spatial Ontology. *Spatial Cognition and Computation*, 4(1):69–104, 2004.
34. Michael R. Hansen. *Logics of Specification Languages*, chapter Duration Calculus, pages 299–347 in [14]. Springer, 2008.
35. Martin C. Henson, Moshe Deutsch, and Steve Reeves. *Logics of Specification Languages*, chapter Z Logic and Its Applications, pages 489–596 in [14]. Springer, 2008.
36. Martin C. Henson, Steve Reeves, and Jonathan P. Bowen. Z Logic and its Consequences. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [58, 17, 21, 56, 31, 53] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
37. Charles Anthony Richard Hoare and Ji Feng He. *Unifying Theories of Programming*. Prentice Hall, 1997.
38. Tony Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.
39. Tony Hoare. Communicating Sequential Processes. Published electronically: <http://www.usingcsp.com/cspbook.pdf>, 2004. Second edition of [38]. See also <http://www.usingcsp.com/>.
40. M. Ingleby. Safety properties of a control network: local and global reasoning in machine proof. In *Proceedings of Real Time Systems*. Paris, January 1994.
41. M. Ingleby. A galois theory of local reasoning in control systems with compositionality. In *Proceedings of Mathematics of Dependable Systems*. Oxford UP (UK), 1995.
42. M. Ingleby and I.H. Mitchell. Proving Safety of a Railway Signaling System Incorporating Geographic Data. In H.H. Frey, editor, *SAFECOM'92 Conference Proceedings of IFAC*, pages 129–134, Zürich (CH), November 1992. Pergamon Press.
43. Daniel Jackson. *Software Abstractions Logic, Language, and Analysis*. The MIT Press, Cambridge, Mass., USA, April 2006. ISBN 0-262-10114-9.
44. Michael A. Jackson. *Software Requirements & Specifications: a lexicon of practice, principles and prejudices*. ACM Press. Addison-Wesley Publishing Company, Wokingham, nr. Reading, England; E-mail: [ipc@awpub.add-wes.co.uk](mailto:ipc@awpub.add-wes.co.uk), 1995. ISBN 0-201-87712-0; xiv + 228 pages.
45. Leslie Lamport. Time, Clcks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, August 1978.

46. C. Lejewski. A note on Leśniewski's axiom system for the mereological notion of ingredient or element. *Topoi*, 2(1):63–71, June, 1983.
47. H.S. Leonard and N. Goodman. The Calculus of Individuals and Its Uses. *Journal of Symbolic Logic*, 5:45–55, 1940.
48. Michael J. Loux. *Metaphysics, a contemporary introduction*. Routledge Contemporary Introductions to Philosophy. Routledge, London and New York, 1998 (2nd ed., 2020).
49. E.C. Luschei. *The Logical Systems of Leśniewski*. North Holland, Amsterdam, The Netherlands, 1962.
50. Z. Manna. *Mathematical Theory of Computation*. McGraw-Hill, 1974.
51. John McCarthy. Towards a Mathematical Science of Computation. In C.M. Popplewell, editor, *IFIP World Congress Proceedings*, pages 21–28, 1962.
52. D. H. Mellor and Alex Oliver. *Properties*. Oxford Readings in Philosophy. Oxford Univ Press, , May 1997. ISBN: 0198751761, 320 pages, Amazon price: US \$ 19.95.
53. Stephan Merz. On the Logic of TLA+. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [58, 17, 21, 56, 31, 36] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
54. Stephan Merz. *Logics of Specification Languages*, chapter The Specification Language TLA<sup>+</sup>, pages 401–451 in [14]. Springer, 2008.
55. T. Mossakowski, A. Haxthausen, D. Sannella, and A. Tarlecki. *Logics of Specification Languages*, chapter CASL – the Common Algebraic Specification Language, pages 241–298 in [14]. Springer, 2008.
56. Till Mossakowski, Anne E. Haxthausen, Don Sanella, and Andrzej Tarlecki. CASL — The Common Algebraic Specification Language: Semantics and Proof Theory. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [58, 17, 21, 31, 53, 36] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
57. Chia-Yi Tony Pi. *Mereology in Event Semantics*. Phd, McGill University, Montreal, Canada, August 1999.
58. Wolfgang Reisig. The Expressive Power of Abstract State Machines. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [17, 21, 56, 31, 53, 36] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
59. Wolfgang Reisig. *Logics of Specification Languages*, chapter Abstract State Machines for the Classroom, pages 15–46 in [14]. Springer, 2008.
60. 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>.
61. Douglas T. Ross. Toward foundations for the understanding of type. In *Proceedings of the 1976 conference on Data : Abstraction, definition and structure*, pages 63–65, New York, NY, USA, 1976. ACM.
62. Stewart Shapiro. *Philosophy of Mathematics — structure and ontology*. Oxford University Press, 1997.
63. Peter M. Simons. *Parts: A Study in Ontology*. Clarendon Press, 1987.
64. J. M. Spivey. *Understanding Z: A Specification Language and its Formal Semantics*, volume 3 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, January 1988.



65. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.
66. J.T.J. Szrednicki and Z. Stachniak, editors. *Leśniewski's Lecture Notes in Logic*. Dordrecht, 1988.
67. Steffen Staab and Rudi Stuber, editors. *Handbook on Ontologies*. International Handbooks on Information Systems. Springer, Heidelberg, 2004.
68. J. C. P. Woodcock and J. Davies. *Using Z: Specification, Proof and Refinement*. Prentice Hall International Series in Computer Science, 1996.

## A Two Axiom Systems for Mereology

### A.1 From: Parts and Places [19]

A mereological system requires at least one primitive binary relation (dyadic predicate). The most conventional choice for such a relation is *Parthood* (also called "inclusion"), "*x* is a part of *y*," written:

$$\mathcal{P}xy$$

Nearly all systems require that *Parthood* partially order the universe. The following defined relations, required for the axioms below, follow immediately from *Parthood* alone:

An immediate defined predicate is "*x* is a proper part of *y*," written  $\mathcal{PP}xy$ , which holds (i.e., is satisfied, comes out true) if  $\mathcal{P}xy$  is true and  $\mathcal{P}yx$  is false. If *Parthood* is a partial order, *ProperPart* is a strict partial order:

$$\mathcal{PP}xy \leftrightarrow (\mathcal{P}xy \wedge \sim \mathcal{P}yx) \quad (1)$$

An object lacking proper parts is an atom. The mereological universe consists of all objects we wish to think about, and all of their proper parts:

*Overlap*: *x* and *y* overlap, written  $\mathcal{O}xy$ , if there exists an object *z* such that  $\mathcal{P}zx$  and  $\mathcal{P}zy$  both hold.

$$\mathcal{O}xy \leftrightarrow \exists z[\mathcal{P}zx \wedge \mathcal{P}zy] \quad (2)$$

The parts of *z*, the "overlap" or "product" of *x* and *y*, are precisely those objects that are parts of both *x* and *y*.

*Underlap*: *x* and *y* underlap, written  $\mathcal{U}xy$ , if there exists an object *z* such that *x* and *y* are both parts of *z*.

$$\mathcal{U}xy \leftrightarrow \exists z[\mathcal{P}xz \wedge \mathcal{P}yz] \quad (3)$$

Overlap and Underlap are reflexive, symmetric, and intransitive.

Systems vary in what relations they take as primitive and as defined. For example, in extensional mereologies (defined below), *Parthood* can be defined from *Overlap* as follows:

$$\mathcal{P}xy \leftrightarrow (\mathcal{O}zx \rightarrow \mathcal{O}zy) \quad (4)$$

### The Axioms

*Parthood* partially orders the universe:

M1, Reflexive: An object is a part of itself.

$$\mathcal{P}xx \quad (5)$$

M2, Antisymmetric: If  $\mathcal{P}xy$  and  $\mathcal{P}yx$  both hold, then  $x$  and  $y$  are the same object.

$$(\mathcal{P}xy \wedge \mathcal{P}yx) \rightarrow x = y \quad (6)$$

M3, Transitive: If  $\mathcal{P}xy$  and  $\mathcal{P}yz$ , then  $\mathcal{P}xz$ .

$$(\mathcal{P}xy \wedge \mathcal{P}yz) \rightarrow \mathcal{P}xz \quad (7)$$

M4, Weak Supplementation: If  $\mathcal{P}\mathcal{P}xy$  holds, there exists a  $z$  such that  $\mathcal{P}zy$  holds but  $\mathcal{O}zx$  does not.

$$\mathcal{P}\mathcal{P}xy \rightarrow \exists z[\mathcal{P}zy \wedge \sim \mathcal{O}zx] \quad (8)$$

M5, Strong Supplementation: Replace “ $\mathcal{P}\mathcal{P}xy$  holds” in M4 with “ $\mathcal{P}yx$  does not hold.”

$$\sim \mathcal{P}yx \rightarrow \exists z[\mathcal{P}zy \wedge \sim \mathcal{O}zx] \quad (9)$$

M5', Atomistic Supplementation: If  $\mathcal{P}xy$  does not hold, then there exists an atom  $z$  such that  $\mathcal{P}zx$  holds but  $\mathcal{O}zy$  does not.

$$\sim \mathcal{P}xy \rightarrow \exists z[\mathcal{P}zx \wedge \sim \mathcal{O}zy \wedge \sim \exists x[\mathcal{P}\mathcal{P}vz]] \quad (10)$$

Top: There exists a “universal object”, designated  $\Omega$ , such that  $\mathcal{P}x\Omega$  holds for any  $x$ .

$$\exists \Omega \forall x[\mathcal{P}x\Omega] \quad (11)$$

Bottom: There exists an atomic “null object”, designated  $\mathcal{U}_{\text{oid}}$ , such that  $\mathcal{P}\mathcal{U}_{\text{oid}}x$  holds for any  $x$ .

$$\exists \mathcal{U}_{\text{oid}} \forall x[\mathcal{P}\mathcal{U}_{\text{oid}}x] \quad (12)$$

M6, Sum: If  $\mathcal{U}xy$  holds, there exists a  $z$ , called the “sum” or “fusion” of  $x$  and  $y$ , such that the parts of  $z$  are just those objects which are parts of either  $x$  or  $y$ .

$$\mathcal{U}xy \rightarrow \exists z \forall v[\mathcal{O}vz \leftrightarrow (\mathcal{O}vx \vee \mathcal{O}vy)] \quad (13)$$

M7, Product: If  $\mathcal{O}xy$  holds, there exists a  $z$ , called the “product” of  $x$  and  $y$ , such that the parts of  $z$  are just those objects which are parts of both  $x$  and  $y$ .

$$\mathcal{O}xy \rightarrow \exists z \forall v[\mathcal{P}vz \leftrightarrow (\mathcal{P}vx \wedge \mathcal{P}vy)] \quad (14)$$

If  $\mathcal{O}xy$  does not hold,  $x$  and  $y$  have no parts in common, and the product of  $x$  and  $y$  is defined iff Bottom holds.

M8, Unrestricted Fusion: Let  $\phi(x)$  be a first-order formula in which  $x$  is a free variable. Then the fusion of all objects satisfying  $\phi$  exists.

$$\exists xz[\phi(x) \rightarrow \forall y[\mathcal{O}yz \leftrightarrow (\phi(x) \wedge \mathcal{O}yx)]] \quad (15)$$

M8', Unique Fusion: The fusions whose existence M8 asserts are also unique.

M9, Atomicity: All objects are either atoms or fusions of atoms.

$$\exists yz[\mathcal{P}yx \wedge \sim \mathcal{P}zy] \quad (16)$$

## A.2 From: A Calculus of Individuals Based on 'Connection' [20]

Taking  $\mathcal{C}xy$  as a rendering of  $x$  is connected to  $y$  we can introduce a definition of  $\mathcal{DC}xy$  ( $x$  is disconnected from  $y$ ) and the standard mereological definitions of  $\mathcal{P}xy$  ( $x$  is a part of  $y$ ),  $\mathcal{PP}xy$  ( $x$  is a proper part of  $y$ ),  $\mathcal{O}xy$  ( $x$  overlaps  $y$ ), and  $\mathcal{DR}xy$  ( $x$  is discrete from  $y$ ) as follows:

$$\mathcal{DC}xy \equiv \sim \mathcal{C}xy \quad (17)$$

$$\mathcal{P}xy \equiv (\forall z)(\mathcal{C}zx \supset \mathcal{C}zy) \quad (18)$$

$$\mathcal{PP}xy \equiv \mathcal{P}xy \wedge \sim \mathcal{P}yx \quad (19)$$

$$\mathcal{O}xy \equiv (\exists z)(\mathcal{P}zx \wedge \mathcal{P}xy) \quad (20)$$

$$\mathcal{DR}xy \equiv \sim \mathcal{O}xy \quad (21)$$

This distinction between  $\mathcal{C}xy$  and  $x$ , constitutes the virtue of this new calculus. It gives us the power to define  $\mathcal{EC}xy$  ( $x$  is externally connected to  $y$ ),  $\mathcal{TP}xy$  ( $x$  is a tangential part of  $y$ ), and  $\mathcal{NTP}xy$  ( $x$  is a nontangential part of  $y$ ) as follows:

$$\mathcal{EC}xy \equiv \mathcal{C}xy \wedge \sim \mathcal{C}xy \quad (22)$$

$$\mathcal{TP}xy \equiv (\forall z)(\mathcal{EC}zx \wedge \mathcal{EC}xy) \quad (23)$$

$$\mathcal{NTP}xy \equiv \mathcal{P}xy \wedge \sim (\forall z)(\mathcal{EC}zx \wedge \mathcal{EC}xy) \quad (24)$$

Our axiomatization requires only two axioms: a mereological axiom,

$$(\forall x)[\mathcal{C}xx \wedge (\forall y)\mathcal{C}xy \supset \mathcal{C}yx] \quad (25)$$

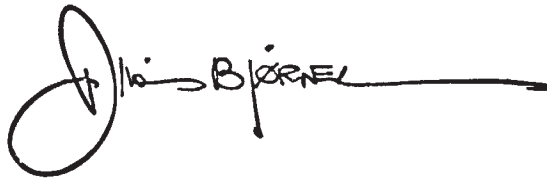
and an axiom involving identity, analogous to the axiom of extension in set theory,

$$(\forall x)(\forall y)[(\forall z)(\mathcal{C}zx = \mathcal{C}zy) \supset x = y]. \quad (26)$$

## B Laudatio

Willem-Paul, how am I, a lowly, modest 'software engineering' researcher, to formulate an appropriate Laudatio, to You, a towering, vocal computer scientist? Our interests are far and wide apart: You delve deeply and successfully into computer science: the study of the "things" that can exist inside computers, I delve into computing science: the study of how to construct those "things". Thanks for Your always vigilant, shameless observance of precision, conciseness, *Etcetera*.

Yet our roads have crossed; many time. And all of these encounters have been delightful. Despite Your throwing of rotten apples, despite Your swinging of mighty swords and despite Your utterings of foul condemnations<sup>16</sup>. Never boring. Sometimes a bit lofty, dispensing, with absolute authority, "wise-man" advice to, well, a bit more experienced people. But a conference without Willem-Paul is not as fun as one with him — and when she's there, and we can't stand Your antics, we can always enjoy Corinne ...




---

<sup>16</sup> The story of the six issues that any conference session chairman should observe was inspired by a WPdR incident at the 1986 IFIP World Computer Congress in Dublin, Ireland:

1. Introduce the speaker on time;
2. "terminate" the speaker on time;
3. ensure that questions are asked;
4. and that questions are answered;
5. protect the audience from abuse from the speaker;
6. and protect the speaker from abuse from the audience.

The last two "rules" are also referred to a "Lex de Roever".