A Rôle for Mereology in Domain Science and Engineering: – to every Mereology there corresponds a λ–expression –

Dines Bjørner
DTU Informatics, Technical University of Denmark
DK-2800 Kgs.Lyngby, Denmark
`bjorner@gmail.com, www.imm.dtu.dk/~dibj`

June 15, 2012

In memory of Douglas T. Ross 1929–2007[1]

Abstract

We give an abstract model of parts and part-hood relations of software application domains such as the financial service industry, railway systems, road transport systems, health care, oil pipelines, secure [IT] systems, etcetera. We relate this model to axiom systems for mereology [6], showing satisfiability, and show that for every mereology there corresponds a class of `Communicating Sequential Processes` [10], that is: a λ–expression.

1 Introduction 1

The term 'mereology' is accredited to the Polish mathematician, philosopher and logician Stansław Leśniewski (1886–1939) who "was a nominalist: he rejected axiomatic set theory and devised three formal systems,*Protothetic*, *Ontology*, and *Mereology* as a concrete alternative to set theory". In this contribution I shall be concerned with only certain aspects of mereology, namely those that appears most immediately relevant to domain science (a relatively new part of current computer science). Our knowledge of 'mereology' has been through studying, amongst others, [6, 11].

1.1 Computing Science Mereology 2

"Mereology (from the Greek $\mu\epsilon\rho o\varsigma$ 'part') is the theory of parthood relations: of the relations of part to whole and the relations of part to part within a whole"[2]. In this contribution we restrict 'parts' to be those that, firstly, are spatially distinguishable, then, secondly, while "being based" on such spatially distinguishable parts, are conceptually related. The relation: "being based", shall be made clear in this contribution. 3

[1] See the big paragraph first in Sect. 7.1.
[2] Achille Varzi: Mereology, http://plato.stanford.edu/entries/mereology/ 2009 and [6]

Accordingly two parts, p_x and p_y, (of a same "whole") are are either "adjacent", or are "embedded within" one another as loosely indicated in Fig. 1.

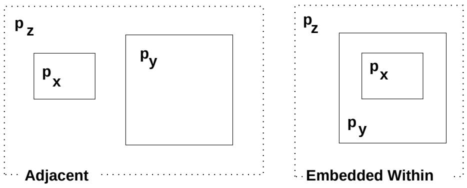


Figure 1: 'Adjacent' and "Embedded Within' parts

4 'Adjacent' parts are direct parts of a same third part, p_z, i.e., p_x and p_y are "embedded within" p_z; or one (p_x) or the other (p_y) or both (p_x and p_y) are parts of a same third part, p'_z "embedded within" p_z; etcetera; as loosely indicated in Fig. 2. or one is "embedded within"
5 the other — etc. as loosely indicated in Fig. 2.

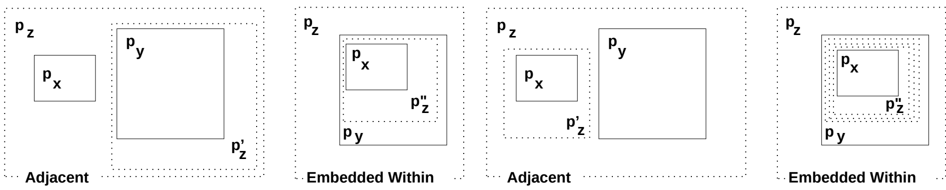


Figure 2: 'Adjacent' and "Embedded Within' parts

Parts, whether adjacent or embedded within one another, can share properties. For adjacent parts this sharing seems, in the literature, to be diagrammatically expressed by letting the part rectangles "intersect". Usually properties are not spatial hence 'intersection' seems confusing.
6 We refer to Fig. 3.

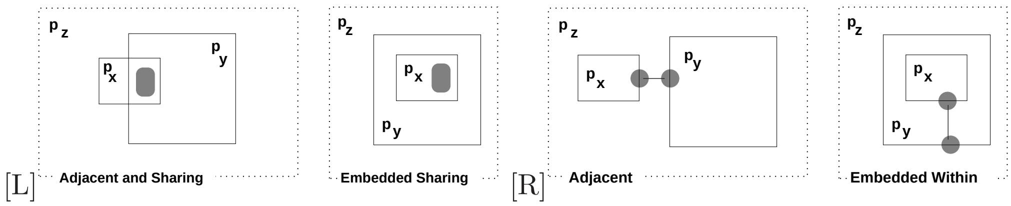


Figure 3: Two models, [L,R], of parts sharing properties

Instead of depicting parts sharing properties as in Fig. 3[L]eft where dashed rounded edge rectangles stands for 'sharing', we shall (eventually) show parts sharing properties as in Fig. 3[R]ight where •—• connections connect those parts.

1.2 From Domains via Requirements to Software 7

One reason for our interest in mereology is that we find that concept relevant to the modelling of domains. A derived reason is that we find the modelling of domains relevant to the develop-

ment of software. Conventionally a first phase of software development is that of requirements engineering. To us domain engineering is (also) a prerequisite for requirements engineering [2, 4]. Thus to properly **design** Software we need to **understand** its or their Requirements; 8
and to properly **prescribe** Requirements one must **understand** its Domain. To **argue** correctness of Software with respect to Requirements one must usually **make assumptions** about the Domain: $\mathbb{D}, \mathbb{S} \models \mathbb{R}$. Thus **description** of Domains become an indispensable part of Software development.

1.3 Domains: Science and Engineering 9

Domain science is the study and knowledge of domains. **Domain engineering** is the practice of **"walking the bridge"** from domain science to domain descriptions: to **create domain descriptions** on the background of scientific knowledge of domains, the specific domain "at hand", or domains in general; and to **study domain descriptions** with a view to broaden and deepen scientific results about domain descriptions. This contribution is based on the engineering and study of many descriptions, of air traffic, banking, commerce (the consumer/retailer/wholesaler/producer supply chain), container lines, health care, logistics, pipelines, railway systems, secure [IT] systems, stock exchanges, etcetera.

1.4 Contributions of This Contribution 10

A general contribution is that of providing elements of a domain science. Three specific contributions are those of (i) giving a model that satisfies published formal, axiomatic characterisations of mereology; (ii) showing that to every (such modelled) mereology there corresponds a `CSP` [10] program and to conjecture the reverse; and, related to (ii), (iii) suggesting complementing **syntactic** and **semantic** theories of mereology.

1.5 Structure of This Contribution 11

We briefly overview the structure of this contribution. First, on Sect. 2, **we loosely characterise how we look at mereologies: "what they are to us !"**. Then, in Sect. 3, **we give an abstract, model-oriented specification of a class of mereologies** in the form of composite parts and composite and atomic subparts and their possible connections. The abstract model as well as the axiom system (Sect. 4) focuses on the **syntax of mereologies.** Following that, 12
in Sect. 4 **we indicate how the model of** Sect. 3 **satisfies the axiom system of that section.** In preparation for Sect. 6, Sect. 5 **presents characterisations of attributes of parts, whether atomic or composite.** Finally Sect. 6 presents **a semantic model of mereologies,** one of a wide variety of such possible models. This one emphasize the possibility of considering parts and subparts as processes and hence a mereology as a system of processes. Section 7 concludes with some remarks on what we have achieved.

2 Our Concept of Mereology 13

2.1 Informal Characterisation

Mereology, to us, is the study and knowledge about how physical and conceptual parts relate and what it means for a part to be related to another part: *being disjoint*, *being adjacent*, *being neighbours*, *being contained properly within*, *being properly overlapped with*, etcetera.

14 By physical parts we mean such spatial individuals which can be pointed to. **Examples:** *a
road net (consisting of street segments and street intersections); a street segment (between
two intersections); a street intersection; a road (of sequentially neigbouring street segments
15 of the same name) a vehicle; and a platoon (of sequentially neigbouring vehicles).*

By a conceptual part we mean an abstraction with no physical extent, which is either
present or not. **Examples:** *a bus timetable (not as a piece or booklet of paper, or as an
electronic device, but) as an image in the minds of potential bus passengers; and routes of a
pipeline, that is, neighbouring sequences of pipes, valves, pumps, forks and joins, for example
referred to in discourse: the gas flows through "such-and-such" a route".* The tricky thing
here is that a route may be thought of as being both a concept or being a physical part —
in which case one ought give them different names: a planned route and an actual road, for
16 example.

The mereological notion of subpart, that is: *contained within* can be illustrated by **exam-
ples:** *the intersections and street segments are subparts of the road net; vehicles are subparts
17 of a platoon; and pipes, valves, pumps, forks and joins are subparts of pipelines.* The mereo-
logical notion of adjacency can be illustrated by **examples.** We consider *the various controls
of an air traffic system, cf. Fig. 4 on the facing page, as well as its aircrafts as adjacent within
the air traffic system; the pipes, valves, forks, joins and pumps of a pipeline, cf. Fig. 9 on
Page 8, as adjacent within the pipeline system; two or more banks of a banking system, cf.
18 Fig. 6 on Page 6, as being adjacent.* The mereo-topological notion of neighbouring can be
illustrated by **examples:** *Some adjacent pipes of a pipeline are neighbouring (connected) to
other pipes or valves or pumps or forks or joins, etcetera; two immediately adjacent vehicles
19 of a platoon are neighbouring.* The mereological notion of proper overlap can be illustrated
by **examples** some of which are of a general kind: *two routes of a pipelines may overlap; and
two conceptual bus timetables may overlap with some, but not all bus line entries being the
same;* and some of really reflect adjacency: *two adjacent pipe overlap in their connection, a
wall between two rooms overlap each of these rooms — that is, the rooms overlap each other
"in the wall".*

2.2 Six Examples 20

We shall, in Sect. 3, present a model that is claimed to abstract essential mereological properties of air traffic, buildings and their installations, machine assemblies, financial service industry, the oil industry and oil pipelines, and railway nets.

2.2.1 Air Traffic 21

Figure 4 on the next page shows nine adjacent (9) boxes and eighteen adjacent (18) lines. Boxes and lines are parts. The line parts "neighbours" the box parts they "connect". Individually boxes and lines represent adjacent parts of the composite air traffic "whole". The rounded corner boxes denote buildings. The sharp corner box denote an aircraft. Lines denote radio telecommunication. The "overlap" between neigbouring line and box parts are indicated by "connectors". Connectors are shown as small filled, narrow, either horisontal or vertical "filled" rectangle[3] at both ends of the double-headed-arrows lines, overlapping both the line arrows and the boxes. The index ranges shown attached to, i.e., labelling each unit, shall indicate that there are a multiple of the "single" (thus representative) box or line unit

[3] There are 38 such rectangles in Fig. 4 on the facing page.

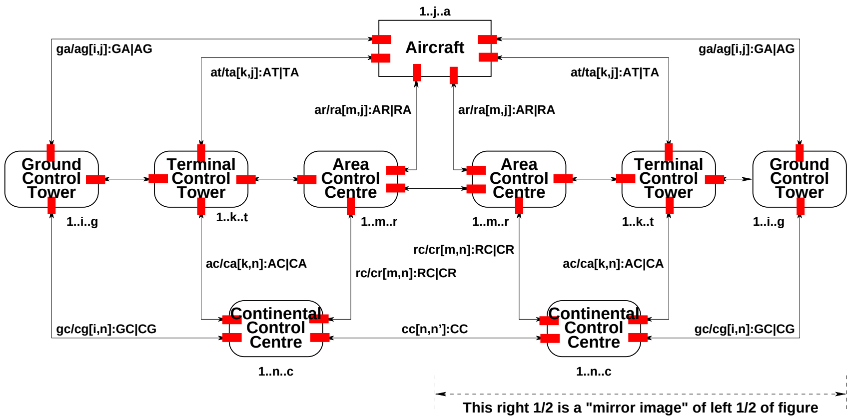


Figure 4: A schematic air traffic system

shown. These index annotations are what makes the diagram of Fig. 4 schematic. Notice that the 'box' parts are fixed installations and that the double-headed arrows designate the ether where radio waves may propagate. We could, for example, assume that each such line is characterised by a combination of location and (possibly encrypted) radio communication frequency. That would allow us to consider all lines for not overlapping. And if they were overlapping, then that must have been a decision of the air traffic system.

2.2.2 Buildings 22

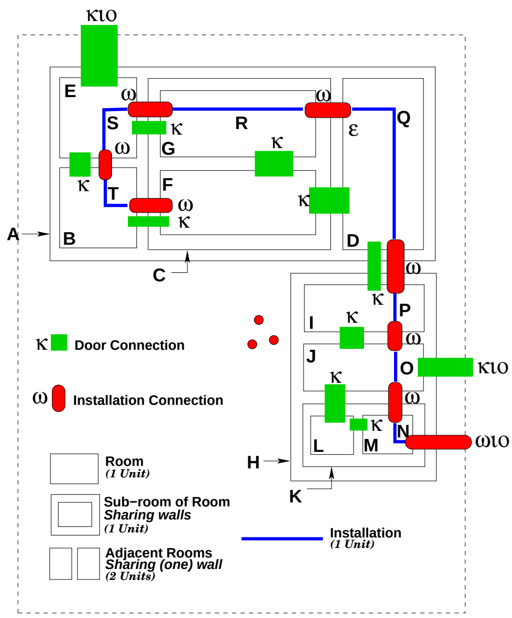


Figure 5: A building plan with installation

Figure 5 shows a building plan — as a composite part. The building consists of two buildings, A and H. The buildings A and H are neighbours, i.e., shares a common wall. Building A has rooms B, C, D and E, Building H has roomsI, J and K; Rooms L and M are within K. Rooms F and G are within C.

The thick lines labelled N, O, P, Q, R, S, and T models either electric cabling, water supply, air conditioning, or some such "flow" of gases or liquids.

Connection $\kappa\iota o$ provides means of a connection between an environment, shown by dashed lines, and B or J, i.e. "models", for example, a door. Connections κ provides "access" between neighbouring rooms. Note that 'neighbouring' is a transitive relation. Connection $\omega\iota o$ allows electricity (or water, or oil) to be conducted between an environment and a room. Connection ω allows electricity (or water, or oil) to be conducted through a wall. Etcetera.

Thus "the whole" consists of A and B. Immediate subparts of A are B, C, D and E. Immediate subparts of C are G and F. Etcetera.

2.2.3 Financial Service Industry 23

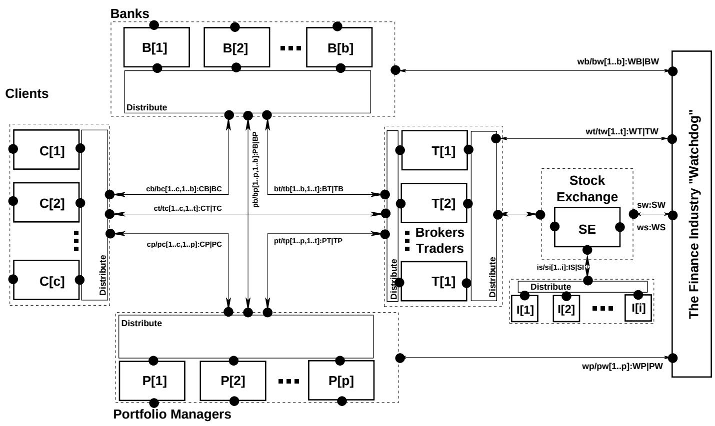


Figure 6: A financial service industry

Figure 6 is rather rough-sketchy! It shows seven (7) larger boxes [6 of which are shown by dashed lines], six [6] thin lined "distribution" boxes, and twelve (12) double-arrowed lines. Boxes and lines are parts. (We do not described what is meant by "distribution".) Where double-arrowed lines touch upon (dashed) boxes we have connections. Six (6) of the boxes, the dashed line boxes, are composite parts, five (5) of them consisting of a variable number of atomic parts; five (5) are here shown as having three atomic parts each with bullets "between" them to designate "variability". Clients, not shown, access the outermost (and hence the "innermost" boxes, but the latter is not shown) through connections, shown by bullets, •.

2.2.4 Machine Assemblies 24

Figure 7 on the facing page shows a machine assembly. Square boxes show composite and atomic parts. Black circles or ovals show connections. The full, i.e., the level 0, composite part consists of four immediate parts and three internal and three external connections. The Pump is an assembly of six (6) immediate parts, five (5) internal connections and three (3)

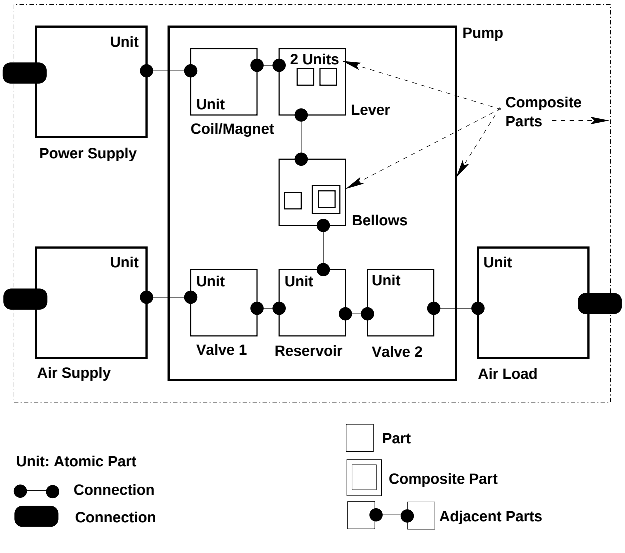


Figure 7: An air pump, i.e., a physical mechanical system

external connectors. Etcetera. Some connections afford "transmission" of electrical power. Other connections convey torque. Two connections convey input air, respectively output air.

2.2.5 Oil Industry

25

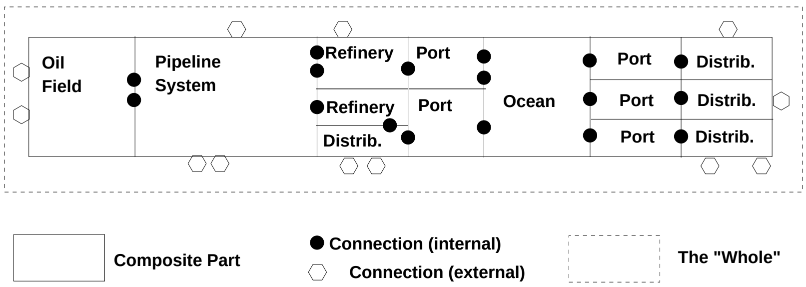


Figure 8: A Schematic of an Oil Industry

"The" Overall Assembly Figure 8 shows a composite part consisting of fourteen (14) composite parts, left-to-right: one oil field, a crude oil pipeline system, two refineries and one, say, gasoline distribution network, two seaports, an ocean (with oil and ethanol tankers and their sea lanes), three (more) seaports, and three, say gasoline and ethanol distribution networks.

Between all of the neighbouring composite parts there are connections, and from some of these composite parts there are connections (to an external environment). The crude oil pipeline system composite part will be concretised next.

26

A Concretised Composite parts Figure 9 on the following page shows a pipeline system. It consists of 32 atomic parts: fifteen (15) pipe units (shown as directed arrows and labelled p1–p15), four (4) input node units (shown as small circles, ◦, and labelled in$i$–inℓ), four (4)

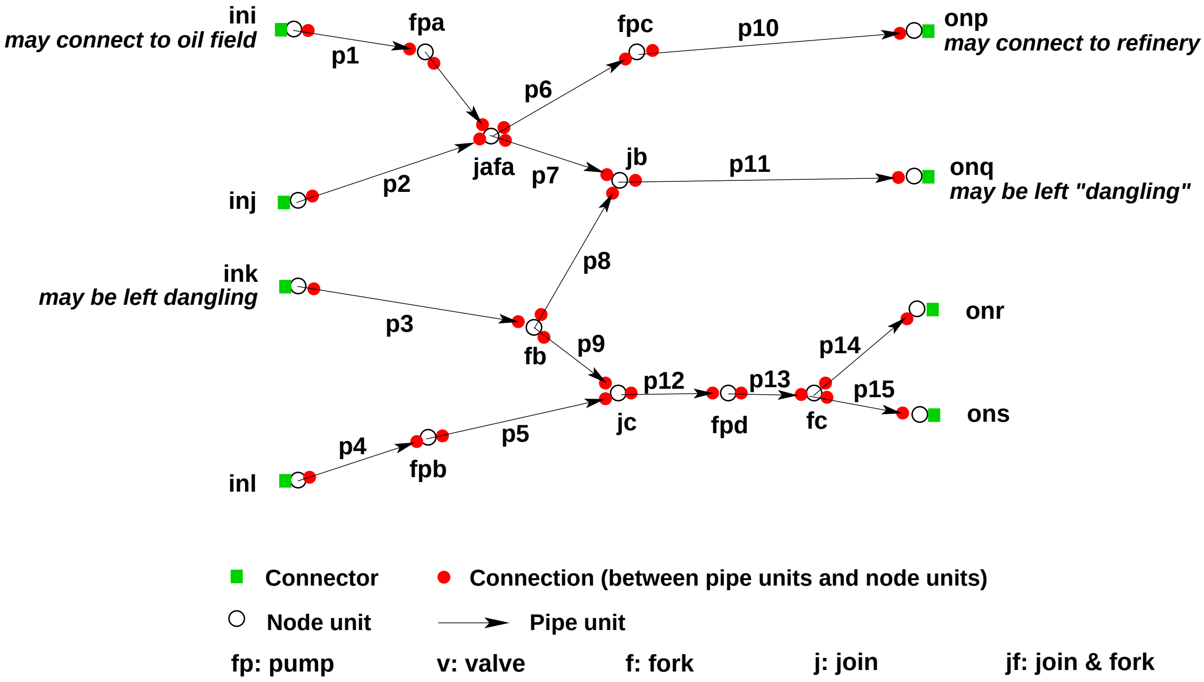


Figure 9: A pipeline system

flow pump units (shown as small circles, ∘, and labelled fp$a$–fp$d$), five (5) valve units (shown as small circles, ∘, and labelled vx–vw), three (3) join units (shown as small circles, ∘, and labelled jb–jc), two (2) fork units (shown as small circles, ∘, and labelled fb–fc), one (1) combined join & fork unit (shown as small circles, ∘, and labelled jf$a$), and four (4) output node units (shown as small circles, ∘, and labelled on$p$–on$s$).

In this example the routes through the pipeline system start with node units and end with node units, alternates between node units and pipe units, and are connected as shown by fully filled-out dark coloured disc connections. Input and output nodes have input, respectively output connections, one each, and shown as lighter coloured connections.

2.2.6 Railway Nets 27

Figure 10 on the next page diagrams four rail units, each with two, three or four connectors shown as narrow, somewhat "longish" rectangles. Multiple instances of these rail units can be assembled (i.e., composed) by their connectors as shown on Fig. 11 on Page 10 into proper
28 rail nets.

Figure 11 on Page 10 diagrams an example of a proper rail net. It is assembled from the kind of units shown in Fig. 10. In Fig. 11 consider just the four dashed boxes: The dashed boxes are assembly units. Two designate stations, two designate lines (tracks) between stations. We refer to to the caption four line text of Fig. 10 on the facing page for more "statistics". We could have chosen to show, instead, for each of the four "dangling' connectors, a composition of a connection, a special "end block" rail unit and a connector.

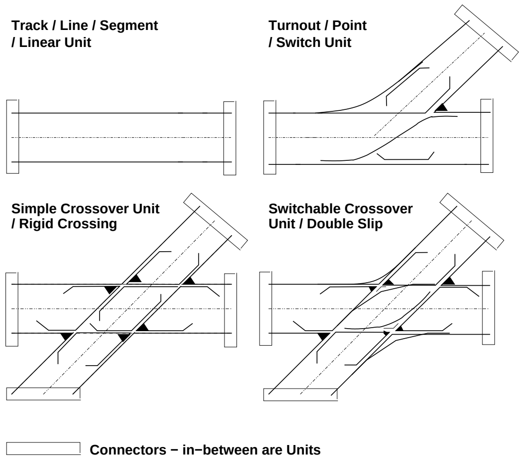


Figure 10: Four example rail units

2.2.7 Discussion 29

We have brought these examples only to indicate the issues of a "whole" and atomic and composite parts, adjacency, within, neighbour and overlap relations, and the ideas of attributes and connections. We shall make the notion of 'connection' more precise in the next section. [17] gives URLs to a number of domain models illustrating a great variety of mereologies.

3 An Abstract, Syntactic Model of Mereologies 30

We distinguish between **atomic** and **composite parts.** Atomic parts do not contain separately distinguishable parts. Composite parts contain at least one separately distinguishable part. It is the domain analyser who decides what constitutes "the whole", that is, how parts relate to one another, what constitutes parts, and whether a part is atomic or composite. We refer to the proper parts of a composite part as subparts.

3.1 Parts and Subparts 31

Figure 12 on Page 11 illustrates composite and atomic parts. The *slanted sans serif* uppercase identifiers of Fig. 12 *A1, A2, A3, A4, A5, A6* and *C1, C2, C3* are meta-linguistic, that is. they stand for the parts they "decorate"; they are not identifiers of "our system".

3.1.1 The Model 32

The formal models of this contribution are expressed in the `RAISE` Specification Language, `RSL` [9, 8, 1].

1. The "whole" contains a set of parts.

2. A part is either an atomic part or a composite part.

3. One can observe whether a part is atomic or composite.

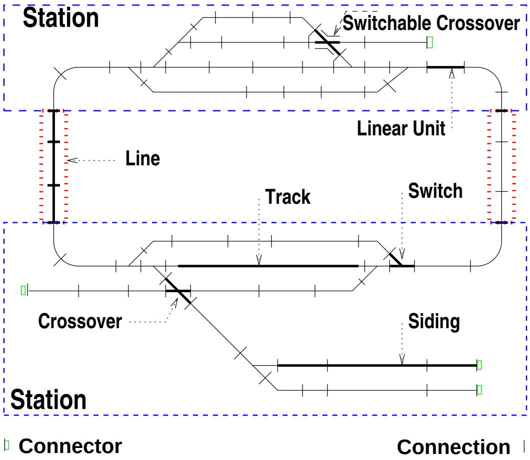


Figure 11: A "model" railway net. An Assembly of four Assemblies: two stations and two lines; Lines here consist of linear rail units; stations of all the kinds of units shown in Fig. 10 on the preceding page. There are 66 connections and four "dangling" connectors

4. Atomic parts cannot be confused with composite parts.

5. From a composite part one can observe one or more parts.

type
1. W = P-**set**
2. P = A | C
value
3. is_A: P $\rightarrow$ **Bool**, is_C: P $\rightarrow$ **Bool**
axiom
4. $\forall$ a:A,c:C•a$\neq$c, i.e., A$\cap$C={||} $\wedge$ is_A(a)$\equiv$∼is_C(a)$\wedge$is_C(c)$\equiv$∼is_A(c)
value
5. obs_Ps: C $\rightarrow$ P-**set** **axiom** $\forall$ c:C • obs_Ps(c)$\neq${}

33 Fig. 12 on the facing page and the expressions below illustrate the observer function obs_Ps:

- obs_Ps(*C1*) = {*A2, A3, C3*},
- obs_Ps(*C2*) = {*A4, A5*},
- obs_Ps(*C3*) = {*A6*}.

34 Please note that this example is meta-linguistic. We can define an auxiliary function.

6. From a composite part, c, we can extract all atomic and composite parts

   a observable from c or

   b extractable from parts observed from c.

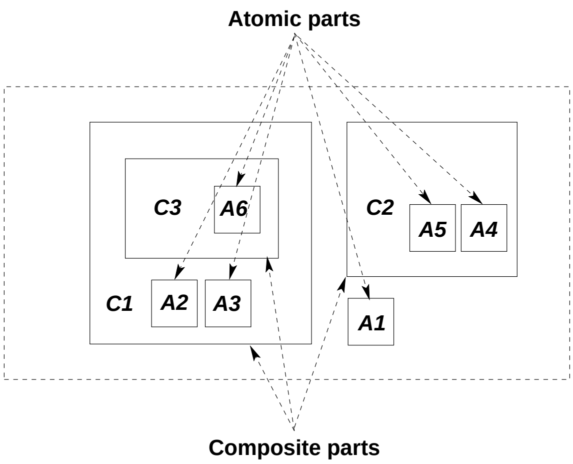


Figure 12: Atomic and composite parts

value
6. xtr_Ps: C → P-**set**
6. xtr_Ps(c) ≡
6a. **let** ps = obs_Ps(c) **in**
6b. ps ∪ ∪ {obs_Ps(c′)|c′:C • c′ ∈ ps} **end**

3.2 'Within' and 'Adjacency' Relations 35

3.2.1 'Within'

7. One part, p, is said to be *immediately within*, imm_within(p,p′), another part,
   a if p′ is a composite part
   b and p is observable in p′.

value
7. imm_within: P × P $\xrightarrow{\sim}$ **Bool**
7. imm_within(p,p′) ≡
7a. is_C(p′)
7b. ∧ p ∈ obs_Ps(p′)

3.2.2 'Transitive Within' 36

We can generalise the 'immediate within' property.

8. A part, p, is transitively within a part p′, within(p,p′),
   a either if p, is immediately within p′
   b or if there exists a (proper) composite part p″ of p′ such that within(p″,p).

value
8. within: $P \times P \xrightarrow{\sim} \textbf{Bool}$
8. within(p,p′) ≡
8a. imm_within(p,p′)
8b. ∨ ∃ p″:C • p″ ∈ obs_Ps(p′) ∧ within(p,p″)

3.2.3 'Adjacency' 37

9. Two parts, p,p′, are said to be *immediately adjacent*, imm_adjacent(p,p′)(c), to one another, in a composite part c, such that p and p′ are distinct and observable in c.

value
9. imm_adjacent: $P \times P \rightarrow C \xrightarrow{\sim} \textbf{Bool}$,
9. imm_adjacent(p,p′)(c) ≡ p≠p′ ∧ {p,p′}⊆obs_Ps(c)

3.2.4 Transitive 'Adjacency' 38

We can generalise the immediate 'adjacent' property.

10. Two parts, p,p′, of a composite part, c, are adjacent(p, p′) in c
    a either if imm_adjacent(p,p′)(c),
    b or if there are two p″ and p‴ of c such that
        i. p″ and p‴ are immediately adjacent parts of c and
        ii. p is equal to p″ or p″ is properly within p and p′ is equal to p‴ or p‴ is properly within p′

value
10. adjacent: $P \times P \rightarrow C \xrightarrow{\sim} \textbf{Bool}$
10. adjacent(p,p′)(c) ≡
10a. imm_adjacent(p,p′)(c) ∨
10b. ∃ p″,p‴:P •
10(b)i. imm_adjacent(p″,p‴)(c) ∧
10(b)ii. ((p=p″)∨within(p,p″)(c)) ∧ ((p′=p‴)∨within(p′,p‴)(c))

3.3 Unique Identifications 39

Each physical part can be uniquely distinguished for example by an abstraction of its properties at a time of origin. In consequence we also endow conceptual parts with unique identifications.

11. In order to refer to specific parts we endow all parts, whether atomic or composite, with **u**nique **id**entifications.

12. We postulate functions which observe these **u**nique **id**entifications, whether as parts in general or as atomic or composite parts in particular.

13. such that any to parts which are distinct have **u**nique **id**entifications.

type
11. Π
value
12. uid_Π: P $\to$ Π
axiom
13. $\forall$ p,p$'$:P • p$\neq$p$'$ $\Rightarrow$ uid_Π(p)$\neq$uid_Π(p$'$)

Figure 13 illustrates the unique identifications of composite and atomic parts. 40

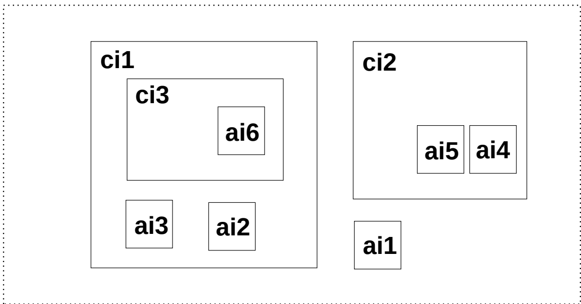


Figure 13: ai_j: atomic part identifiers, ci_k: composite part identifiers

We exemplify the observer function obs_Π in the expressions below and on Fig. 13: 41

- obs_Π($C1$) = $ci1$, obs_Π($C2$) = $ci2$, etcetera; and
- obs_Π($A1$) = $ai1$, obs_Π($A2$) = $ai2$, etcetera.

Please note that also this example is meta-linguistic. 42

14. We can define an auxiliary function which extracts all part identifiers of a composite part and parts within it.

value
14. xtr_Πs: C $\to$ Π-**set**
14. xtr_Πs(c) $\equiv$ {uid_Π(c)} $\cup$ $\cup$ {uid_Π(p)|p:P•p $\in$ xtr_Πs(c)}

3.4 Attributes 43

In Sect. 5 we shall explain the concept of properties of parts, or, as we shall refer to them, attributes For now we just postulate that

15. parts have sets of attributes, atr:ATR, (whatever they are!),
16. that we can observe attributes from parts, and hence
17. that two distinct parts may share attributes

18. for which we postulate a membership function ∈.

type
15. ATR
value
16. atr_ATRs: P → ATR-**set**
17. share: P×P → **Bool**
17. share(p,p′) ≡ p≠p′∧∃ atr:ATR•atr ∈ atr_ATRs(p)∧atr ∈ atr_ATRs(p′)
18. ∈: ATR × ATR-**set** → **Bool**

3.5 Connections 44

In order to illustrate other than the within and adjacency part relations we introduce the notions of connectors and, hence, connections. Figure 14 illustrates connections between
45 parts. A connector is, visually, a •—• line that connects two distinct part boxes.

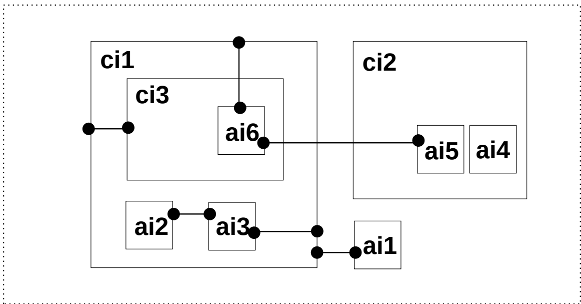


Figure 14: Connectors

46

19. We may refer to the connectors by the two element sets of the unique identifiers of the parts they connect.

    For **example:**

    - $\{ci_1, ci_3\}$,
    - $\{ai_2, ai_3\}$,
    - $\{ai_6, ci_1\}$,
    - $\{ai_3, ci_1\}$,
    - $\{ai_6, ai_5\}$ and
    - $\{ai_1, ci_1\}$.

47

20. From a part one can observe the unique identities of the other parts to which it is connected.

type
19. K = {| k:Π-**set** • **card** k = 2 |}
value
20. mereo_Ks: P → K-**set**

21. The set of all possible connectors of a part can be calculated.

value
21. $\text{xtr\_Ks}: P \to \text{K-}\textbf{set}$
21. $\text{xtr\_Ks}(p) \equiv \{\{\text{uid\_}\Pi(p),\pi\} | \pi{:}\Pi \bullet \pi \in \text{mereo\_}\Pi\text{s}(p)\}$

3.5.1 Connector Wellformedness 48

22. For a composite part, s:C,
23. all the observable connectors, ks,
24. must have their two-sets of part identifiers identify parts of the system.

value
22. $\text{wf\_Ks}: C \to \textbf{Bool}$
22. $\text{wf\_Ks}(c) \equiv$
23. $\quad \textbf{let}\ ks = \text{xtr\_Ks}(c),\ \pi s = \text{mereo\_}\Pi\text{s}(c)\ \textbf{in}$
24. $\quad \forall \{\pi',\pi''\}{:}\Pi\textbf{-set} \bullet \{\pi',\pi''\} \subseteq ks \Rightarrow$
24. $\qquad \exists\ p',p''{:}P \bullet \{\pi',\pi''\} = \{\text{uid\_}\Pi(p'),\text{uid\_}\Pi(p'')\}\ \textbf{end}$

3.5.2 Connector and Attribute Sharing Axioms 49

25. We postulate the following axiom:
    a If two parts share attributes, then there is a connector between them; and
    b if there is a connector between two parts, then they share attributes.
26. The function xtr_Ks (Item 21 on the preceding page) can be extended to apply to Wholes.

axiom
25. $\forall\ w{:}W \bullet$
25. $\quad \textbf{let}\ ps = \text{xtr\_Ps}(w),\ ks = \text{xtr\_Ks}(w)\ \textbf{in}$
25a. $\quad \forall\ p,p'{:}P \bullet p \neq p' \wedge \{p,p'\} \subseteq ps \wedge \text{share}(p,p') \Rightarrow$
25a. $\qquad \{\text{uid\_}\Pi(p),\text{uid\_}\Pi(p')\} \in ks\ \wedge$
25b. $\quad \forall\ \{uid,uid'\} \in ks \Rightarrow$
25b. $\qquad \exists\ p,p'{:}P \bullet \{p,p'\} \subseteq ps \wedge \{uid,uid'\} = \{\text{uid\_}\Pi(p),\text{uid\_}\Pi(p')\}$
25b. $\qquad\quad \Rightarrow \text{share}(p,p')\ \textbf{end}$

value
26. $\text{xtr\_Ks}: W \to \text{K-}\textbf{set}$
26. $\text{xtr\_Ks}(w) \equiv \cup\{\text{xtr\_Ks}(p) | p{:}P \bullet p \in \text{obs\_Ps}(p)\}$

In other words: modelling sharing by means of intersection of attributes or by means of connectors is “equivalent”.

3.5.3 Sharing 50

27. When two distinct parts share attributes,
28. then they are said to be sharing:

27. sharing: $\mathrm{P} \times \mathrm{P} \rightarrow \mathbf{Bool}$
28. sharing(p,p$'$) $\equiv \mathrm{p} \neq \mathrm{p}' \wedge \mathrm{share}(\mathrm{p},\mathrm{p}')$

3.6 Uniqueness of Parts 51

There is one property of the model of wholes: W, Item 1 on Page 9, and hence the model of composite and atomic parts and their unique identifiers "spun off" from W (Item 2 [Page 9] to Item 25b [Page 15]). and that is that any two parts as revealed in different, say adjacent parts are indeed unique, where we — simplifying — define uniqueness sôlely by the uniqueness of their identifiers.

3.6.1 Uniqueness of Embedded and Adjacent Parts 52

29. By the definition of the obs_Ps function, as applied obs_Ps(c) to composite parts, c:C, the atomic and composite subparts of c are all distinct and have distinct identifiers (uiids: unique immediate identifiers).

value
29. uiids: $\mathrm{C} \rightarrow \mathbf{Bool}$
29. uiids(c) $\equiv \forall\ \mathrm{p,p'}:\mathrm{P}\bullet \mathrm{p}\neq\mathrm{p}' \wedge \{\mathrm{p,p'}\}\subseteq \mathrm{obs\_Ps(c)} \Rightarrow \mathbf{card}\{\mathrm{uid\Pi(p)},\mathrm{uid\Pi(p')},\mathrm{uid\Pi(c)}\}=3$

53

30. We must now specify that that uniqueness is "propagated" to parts that are proper parts of parts of a composite part (uids: unique identifiers).

30. uids: $\mathrm{C} \rightarrow \mathbf{Bool}$
30. uids(c) $\equiv$
30. $\quad \forall\ \mathrm{c}':\mathrm{C}\bullet \mathrm{c}' \in \mathrm{obs\_Ps(c)} \Rightarrow \mathrm{uiids(c')}$
30. $\wedge$ **let** $\mathrm{ps}'=\mathrm{xtr\_Ps(c')},\mathrm{ps}''=\mathrm{xtr\_Ps(c'')}$ **in**
30. $\quad \forall\ \mathrm{c}'':\mathrm{C}\bullet \mathrm{c}'' \in \mathrm{ps}' \Rightarrow \mathrm{uids(c'')}$
30. $\wedge\ \forall\ \mathrm{p}',\mathrm{p}'':\mathrm{P}\bullet \mathrm{p}' \in \mathrm{ps}' \wedge \mathrm{p}'' \in \mathrm{ps}'' \Rightarrow \mathrm{uid\_\Pi(p')} \neq \mathrm{uid\_\Pi(p'')}$ **end**

4 An Axiom System 54

Classical axiom systems for mereology focus on just one sort of "things", namely $\mathcal{P}$arts. Leśniewski had in mind, when setting up his mereology to have it supplant set theory. So parts could be composite and consisting of other, the sub-parts — some of which would be atomic; just as sets could consist of elements which were sets — some of which would be empty.

4.1 Parts and Attributes 55

In our axiom system for mereology we shall avail ourselves of two sorts: $\mathcal{P}$arts, and $\mathcal{A}$ttributes.[4]

- **type** $\mathcal{P}, \mathcal{A}$

$\mathcal{A}$ttributes are associated with $\mathcal{P}$arts. We do not say very much about attributes: We think of attributes of parts to form possibly empty sets. So we postulate a primitive predicate, $\in$, relating $\mathcal{P}$arts and $\mathcal{A}$ttributes.

- $\in: \mathcal{A} \times \mathcal{P} \rightarrow$ **Bool**.

4.2 The Axioms 56

The axiom system to be developed in this section is a variant of that in [6]. We introduce the following relations between parts:

part_of:	$\mathbb{P}$:	$\mathcal{P} \times \mathcal{P}$	$\rightarrow$	**Bool**	Page 17
proper_part_of:	$\mathbb{PP}$:	$\mathcal{P} \times \mathcal{P}$	$\rightarrow$	**Bool**	Page 17
overlap:	$\mathbb{O}$:	$\mathcal{P} \times \mathcal{P}$	$\rightarrow$	**Bool**	Page 17
underlap:	$\mathbb{U}$:	$\mathcal{P} \times \mathcal{P}$	$\rightarrow$	**Bool**	Page 18
over_crossing:	$\mathbb{OX}$:	$\mathcal{P} \times \mathcal{P}$	$\rightarrow$	**Bool**	Page 18
under_crossing:	$\mathbb{UX}$:	$\mathcal{P} \times \mathcal{P}$	$\rightarrow$	**Bool**	Page 18
proper_overlap:	$\mathbb{PO}$:	$\mathcal{P} \times \mathcal{P}$	$\rightarrow$	**Bool**	Page 18
proper_underlap:	$\mathbb{PU}$:	$\mathcal{P} \times \mathcal{P}$	$\rightarrow$	**Bool**	Page 18

57

Let $\mathbb{P}$ denote **part-hood**; p_x is part of p_y, is then expressed as $\mathbb{P}(p_x, p_y)$.[5] (1) Part p_x is part of itself (reflexivity). (2) If a part p_x is part p_y and, vice versa, part p_y is part of p_x, then $p_x = p_y$ (antisymmetry). (3) If a part p_x is part of p_y and part p_y is part of p_z, then p_x is part of p_z (transitivity).

$$\forall p_x : \mathcal{P} \bullet \mathbb{P}(p_x, p_x) \tag{1}$$

$$\forall p_x, p_y : \mathcal{P} \bullet (\mathbb{P}(p_x, p_y) \wedge \mathbb{P}(p_y, p_x)) \Rightarrow p_x = p_y \tag{2}$$

$$\forall p_x, p_y, p_z : \mathcal{P} \bullet (\mathbb{P}(p_x, p_y) \wedge \mathbb{P}(p_y, p_z)) \Rightarrow \mathbb{P}(p_z, p_z) \tag{3}$$

58

Let $\mathbb{PP}$ denote **proper part-hood**. p_x is a proper part of p_y is then expressed as $\mathbb{PP}(p_x, p_y)$. $\mathbb{PP}$ can be defined in terms of $\mathbb{P}$. $\mathbb{PP}(p_x, p_y)$ holds if p_x is part of p_y, but p_y is not part of p_x.

$$\mathbb{PP}(p_x, p_y) \triangleq \mathbb{P}(p_x, p_y) \wedge \neg\mathbb{P}(p_y, p_x) \tag{4}$$

59

Overlap, $\mathbb{O}$, expresses a relation between parts. Two parts are said to overlap if they have "something" in common. In classical mereology that 'something' is parts. To us parts are spatial entities and these cannot "overlap". Instead they can 'share' attributes.

$$\mathbb{O}(p_x, p_y) \triangleq \exists a : \mathcal{A} \bullet a \in p_x \wedge a \in p_y \tag{5}$$

60

[4] Identifiers P and A stand for model-oriented types (parts and atomic parts), whereas identifiers $\mathcal{P}$ and $\mathcal{A}$ stand for property-oriented types (parts and attributes).

[5] Our notation now is not `RSL` but a conventional first-order predicate logic notation.

Underlap, $\mathbb{U}$, expresses a relation between parts. Two parts are said to underlap if there exists a part p_z of which p_x is a part and of which p_y is a part.

$$\mathbb{U}(p_x, p_y) \triangleq \exists p_z : \mathcal{P} \bullet \mathbb{P}(p_x, p_z) \wedge \mathbb{P}(p_y, p_z) \tag{6}$$

61 Think of the underlap p_z as an "umbrella" which both p_x and p_y are "under".

Over-cross, $\mathbb{OX}$, p_x and p_y are said to over-cross if p_x and p_y overlap and p_x is not part of p_y.

$$\mathbb{OX}(p_x, p_y) \triangleq \mathbb{O}(p_x, p_y) \wedge \neg\mathbb{P}(p_x, p_y) \tag{7}$$

62

Under-cross, $\mathbb{UX}$, p_x and p_y are said to under cross if p_x and p_y underlap and p_y is not part of p_x.

$$\mathbb{UX}(p_x, p_y) \triangleq \mathbb{U}(p_x, p_z) \wedge \neg\mathbb{P}(p_y, p_x) \tag{8}$$

63

Proper Overlap, $\mathbb{PO}$, expresses a relation between parts. p_x and p_y are said to properly overlap if p_x and p_y over-cross and if p_y and p_x over-cross.

$$\mathbb{PO}(p_x, p_y) \triangleq \mathbb{OX}(p_x, p_y) \wedge \mathbb{OX}(p_y, p_x) \tag{9}$$

64

Proper Underlap, $\mathbb{PU}$, p_x and p_y are said to properly underlap if p_x and p_y under-cross and p_x and p_y under-cross.

$$\mathbb{PU}(p_x, p_y) \triangleq \mathbb{UX}(p_x, p_y) \wedge \mathbb{UX}(p_y, p_x) \tag{10}$$

4.3 Satisfaction 65

We shall sketch a proof that the *model* of the previous section, Sect. 3, *satisfies* is a model for the *axioms* of this section. To that end we first define the notions of *interpretation*,
66 *satisfiability*, *validity* and *model*.

Interpretation: By an interpretation of a predicate we mean an assignment of a truth value to the predicate where the assignment may entail an assignment of values, in general, to the terms of the predicate.

Satisfiability: By the satisfiability of a predicate we mean that the predicate is true for some interpretation.

Valid: By the validity of a predicate we mean that the predicate is true for all interpretations.

Model: By a model of a predicate we mean an interpretation for which the predicate holds.

4.3.1 A Proof Sketch 67

We assign

31. P as the meaning of $\mathcal{P}$
32. ATR as the meaning of $\mathcal{A}$,

33. imm_within as the meaning of $\mathbb{P}$,
34. within as the meaning of $\mathbb{PP}$,
35. $\in_{(\text{of type:}\mathcal{ATR}\times\mathcal{ATR}-\textbf{set}\to\textbf{Bool})}$ as the meaning of $\in_{(\text{of type:}\mathcal{A}\times\mathcal{P}\to\textbf{Bool})}$ and
36. sharing as the meaning of $\mathbb{O}$.

With the above assignments is is now easy to prove that the other axiom-operators $\mathbb{U}$, $\mathbb{PO}$, $\mathbb{PU}$, $\mathbb{OX}$ and $\mathbb{UX}$ can be modelled by means of imm_within, within, $\in_{(\text{of type:}\mathcal{ATR}\times\mathcal{ATR}-\textbf{set}\to\textbf{Bool})}$ and sharing.

5 An Analysis of Properties of Parts 68

So far we have not said much about *"the nature"* of parts other than composite parts having one or more subparts and parts having attributes. In preparation also for the next section, Sect. 6 we now take a closer look at the concept of 'attributes'. We consider three kinds of attributes: their unique identifications [uid_Π] — which we have already considered; their connections, i.e., their mereology [mereo_P] — which we also considered; and their "other" attributes which we shall refer to as properties. [prop_P]

5.1 Mereological Properties 69

5.1.1 An Example

Road nets, n:N, consists of a set of street intersections (hubs), h:H, uniquely identified by hi's (in HI), and a set of street segments (links), l:L, uniquely identified by li's (in LI). such that from a street segment one can observe a two element set of street intersection identifiers, and from a street intersection one can observe a set of street segment identifiers. Constraints 70
between values of link and hub identifiers must be satisfied. The two element set of street intersection identifiers express that the street segment is connected to exactly two existing and distinct street intersections, and the zero, one or more element set of street segment identifiers express that the street intersection is connected to zero, one or more existing and distinct street segments. An axiom expresses these constraints. We call the hub identifiers of hubs and links, the link identifiers of links and hubs, and their fulfilment of the axiom the connection **mereo**logy. 71

```
type
    N, H, L, HI, LI
value
    obs_Hs: N→H-set, obs_Ls: N→L-set
    uid_HI: H→HI, uid_LI: L→LI
    mereo_HIs: L→HI-set axiom ∀ l:L•card mereo_HIs(l)=2
    mereo_LIs: H→LI-set
axiom
  ∀ n:N•
      let hs=obs_Hs(n),ls=obs_Ls(n) in
      ∀ h:H•h ∈ hs ⇒ ∀ li:LI•li ∈ mereo_LIs(h)⇒∃ l:L•uid_LI(l)=li
      ∧ ∀ l:L•l ∈ ls ⇒ ∃ h,h′:H•{h,h′}⊆hs∧mereo_HIs(l)={uid_HI(h),uid_HI(h′)}
      end
```
•

5.1.2 Unique Identifier and Mereology Types 72

In general we allow for any embedded (within) part to be connected to any other embedded part of a composite part or across adjacent composite parts. Thus we must, in general, allow for a family of part types P1, P2, ..., Pn, for a corresponding family of part identifier types Π1, Π2, ..., Πn, and for corresponding observer **u**nique **id**entification and **mer**eology functions:

> **type**
> P = P1 | P2 | ... | Pn
> Π = Π1 | Π2 | ... | Πn
> **value**
> uid_Πj: Pj → Πj for 1≤j≤n
> mereo_Πs: P → Π-**set**

73 **Example:** Our example relates to the abstract model of Sect. 3.

37. With each part we associate a unique identifier, π.
38. And with each part we associate a set, $\{\pi_1, \pi_2, \ldots, \pi_n\}, n \leq 0$ of zero, one ore more other unique identifiers, different from π.
39. Thus with each part we can associate a set of zero, one or more connections, viz.: $\{\pi, \pi_j\}$ for $0 \leq j \leq n$.

74

type
37. Π
value
37. uid_Π: P → Π
38. mereo_Πs: P → Π-**set**
axiom
38. ∀ p:P•uid_Π(p) ∉ mereo_Πs(p)
value
39. xtr_Ks: P → K-**set**
39. xtr_Ks(p) ≡
39. **let** (π,πs)=(uid_Π,mereo_Πs)(p) **in**
39. {{π′,π″}|π′,π″:Π•π′=π∧π″∈πs} **end**

•

5.2 Properties 75

By the properties of a part we mean such properties additional to those of unique identification and mereology. Perhaps this is a cryptic characterisation. Parts, whether atomic or composite, are there for a purpose. The unique identifications and mereologies of parts are there to refer to and structure (i.e., relate) the parts. So they are there to facilitate the purpose. The properties of parts help towards giving these parts "their final meaning". (We shall support
76 his claim ("their final meaning") in Sect. 6.) Let us illustrate the concept of properties.

Examples: (i) Typical properties of street segments are: length, cartographic location, surface material, surface condition, traffic state — whether open in one, the other, both or

77 closed in all directions. (ii) Typical properties of street intersections are: design[6] location, surface material, surface condition, traffic state — open or closed between any two pairs of in/out street segments. (iii) Typical properties of road nets are: name, owner, public/private, free/tool road, area, etcetera. • 78

40. Parts are characterised (also) by a set of one or more distinctly named and not necessarily distinctly typed property values.
    a Property names are further undefined tokens (i.e., simple quantities).
    b Property types are either sorts or are concrete types such as integers, reals, truth values, enumerated simple tokens, or are structured (sets, Cartesians, lists, maps) or are functional types.
    c From a part
        i. one can observe its sets of property names
        ii. and its set (i.e., enumerable map) of distinctly named and typed property values.
    d Given an property name of a part one can observe the value of that part for that property name.
    e For practical reasons we suggest **prop**erty named **prop**erty value observer function — where we further take the liberty of using the **prop**erty type name in lieu of the **prop**erty name.

79

type
40. Props = PropNam $\overrightarrow{m}$ PropVAL
40a. PropNam
40b. PropVAL
value
40(c)i. obs_Props: P $\rightarrow$ Props
40(c)ii. xtr_PropNams: P $\rightarrow$ PropNam-**set**
40(c)ii. xtr_PropNams(p) $\equiv$ **dom** obs_Props(p)
40d. xtr_PropVAL: P $\rightarrow$ PropNam $\xrightarrow{\sim}$ PropVAL
40d. xtr_PropVAL(p)(pn) $\equiv$ (obs_Props(p))(pn)
40d. **pre**: pn $\in$ xtr_PropNams(p)

Here we leave PropNames and PropVALues undefined. 80

Example:

type
NAME, OWNER, LEN, DESIGN, PP == public | private, ...
LΣ, HΣ, LΩ, HΩ
value
obs_Props: N $\rightarrow$ {| ["name"$\mapsto$nm,"owner"$\mapsto$ow,"public/private"$\mapsto$pp,...]
| nm:NAME, ow:OWNER, ..., pp:PP |}

[6] for example, a simple 'carrefour', or a (circular) roundabout, or a free-way interchange a cloverleaf or a stack or a clover-stack or a turbine or a roundabout or a trumpet or a directional or a full Y or a hybrid interchange.

obs_Props: L → {| [″length″↦len,...,″state″↦lσ,″state space″↦lω:LΩ]
| **len**:LEN,...,lσ:LΣ,lω:LΩ |}
obs_Props: H → {| [″design″↦des, ...,″state″↦hσ,″state space″↦hω]
| des:DESIGN,...,hσ:HΣ,hω:HΩ |}
prop_NAME: N → NAME
prop_OWNER: N → OWNER
prop_LEN: L → LEN
prop_LΣ: L → LΣ, obs_LΩ: L → LΩ
prop_DESIGN: H → DESIGN
prop_HΣ: H → HΣ, obs_HΩ: H → HΩ
...

We trust that the reader can decipher this example. •

5.3 Attributes

81

There are (thus) three kinds of part attributes:

- **un**ique **id**entifier "observers" (uid_),
- **mereo**logy "observers (**mereo_**), and
- **prop**erty "observers" (**prop_...**, obs_Props)

We refer to Sect. 3.4, and to Items 15–16.

type
15.′ ATR = Π × Π-**set** × Props
value
16.′ atr_ATR: P → ATR
axiom
∀ p:P • **let** (π,πs,props) = atr_ATR(p) **in** π ∉ πs **end**

82

In preparation for redefining the share function of Item 17 on Page 13 we must first introduce a modification to property values.

41. A property value, pv:PropVal, is either a simple property value (as was hitherto assumed), or is a unique part identifier.

type
40. Props = PropNam $\overrightarrow{m}$ PropVAL_or_Π
41. PropVAL_or_Π :: mk_Simp:PropVAL | mk_Π:Π

83

42. The idea a property name pn, of a part p', designating a Π-valued property value π is
   a that π refers to a part p'
   b one of whose property names must be pn
   c and whose corresponding property value must be a proper, i.e., simple property value, v,
   d which is then the property value in p' for pn.

84

value
42. get_VAL: P × PropName → W → PropVAL
42. get_VAL(p,pn)(w) ≡
44. **let** pv = (obs_Props(p))(pn) **in**

```
42.    case pv of
42.       mk_Simp(v) → v,
42a.       mk_Π(π) →
42a.          let p′:P•p′ ∈ xtr_Ps(w)∧uid_Π(p′)=π in
42c.          (obs_Props(p′))(pn) end
42.    end end
42c.    pre: pn ∈ obs_PropNams(p)
42b.       ∧ pn ∈ obs_PropNams(p′)
42c.       ∧ is_PropVAL((obs_Props(p′))(pn))
```

The three bottom lines above, Items 42b–42c, imply the general constraint now formulated. 85

43. We now express a constraint on our modelling of attributes.
    a Let the attributes of a part p be $(\pi, \pi s, \mathsf{props})$.
    b If a property name pn in props has (associates to) a Π value, say π'
    c then π' must be in πs.
    d and there must exist another part, p', distinct from p, with unique identifier π', such that
    e it has some property named pn with a simple property value.

value

```
43.  wf_ATR: ATR → W → Bool
43a.  wf_ATR(π,πs,props)(w) ≡
43a.     π ∉ πs ∧
43b.     ∀ π′:Π • π′ ∈ rng props ⇒
43c.        let pn:PropNam•props(pn)=π′ in
43c.        pi′∈ πs
43d.      ∧ ∃ p′:P•p′∈ xtr_Ps(w)∧uid_Π(p′)=π′ ⇒
43e.           pn ∈ obs_PropNams(obs_Props(p′))
43e.         ∧ ∃ mk_SimpVAL(v):VAL•(obs_Props(p′))(pn)=mk_SimpVAL(v) end
```

86

44. Two distinct parts share attributes
    a if the unique part identifier of one of the parts is in the mereology of the other part, or
    b if a property value of one of the parts refers to a property of the other part.

87

value

```
44.    share: P × P → Bool
44.    share(p,p′) ≡
44.       p ≠ p′ ∧
44.       let (π,πs,props) = atr_ATR(p),(π′,πs′,props′) = atr_ATR(p′),
44.            pns = xtr_PropNams(p), pns′ = xtr_PropNams(p′) in
44a.     π ∈ πs′ ∨ π′ ∈ πs ∨
44b.     ∃ pn:PropNam•pn ∈ pns ∩ pns′ ⇒
44b.        let vop = props(pn), vop′ = props′(pn) in
44b.        case (vop,vop′) of
44b.           (mk_Π(π″),mk_Simp(v)) → π″=π′,
44b.           (mk_Simp(v),mk_Π(π″)) → π=π″,
44b.           _ → false
44.       end end end
```

Comment: v is a shared attribute.

5.4 Discussion 88

We have now witnessed four kinds of observer function:

- he above three kinds of mereology and property 'observers' and the
- part (and subpart) **obs_**ervers.

These observer functions are postulated. They cannot be defined. They "just exist" by the force of
89 our ability to observe and decide upon their values when applied by us, the domain observers.

Parts are either composite or atomic. Analytic functions are postulated. They help us decide whether a part is composite or atomic, and, from composite parts their immediate subparts.

Both atomic and composite parts have all three kinds of attributes: unique identification, mereology (connections), and properties. Analytic functions help us observe, from a part, its unique
90 identification, its mereology, and its properties.

Some attribute values may be static, that is, constant, others may be inert dynamic, that is, can be changed. It is exactly the inert dynamic attributes which are the basis for the next sections semantic model of parts as processes.

In the above model (of this and Sect. 3) we have not modelled distinctions between static and dynamic properties. You may think, instead of such a model, that an **always** temporal operator, $\Box$, being applied to appropriate predicates.

6 A Semantic CSP Model of Mereology 91

The model of Sect. 3 can be said to be an abstract model-oriented definition of the syntax of mereology. Similarly the axiom system of Sect. 4 can be said to be an abstract property-oriented definition of the syntax of mereology. With the analysis of attributes of parts, Sect. 5, we have begun a semantic analysis of mereology. We now bring that semantic analysis a step further.

6.1 A Semantic Model of a Class of Mereologies 92

We show that to every mereology there corresponds a program of cooperating sequential processes CSP. We assume that the reader has practical knowledge of Hoare's CSP [10].

6.1.1 Parts $\simeq$ Processes

The model of mereology presented in Sect. 3 (Pages 9–16) focused on (i) parts and (ii) connectors. To parts we associate CSP processes. Part processes are indexed by the unique part identifiers. The connectors form the mereological attributes of the model.

6.1.2 Connectors $\simeq$ Channels 93

The CSP channels are indexed by the two-set (hence distinct) part identifier connectors. From a whole we can extract (xtr_Ks, Item 26 on Page 15) all connectors. They become indexes into an array of
94 channels. Each of the connector channel index identifiers indexes exactly two part processes. Let w:W be the whole under analysis.

value
  w:W
  ps:P-**set** $=$ $\cup$\{xtr_Ps(c)|c:C•c $\in$ w\} $\cup$ \{a|a:A•a $\in$ w\}
  ks:K-**set** $=$ xtr_Ks(w)
type
  K $=$ Π-**set axiom** $\forall$ k:K•**card** k=2
  ChMap $=$ Π $\overrightarrow{m}$ K-**set**
value

$\quad$ cm:ChMap = [uid_Π(p)↦xtr_Ks(p)|p:P•p ∈ ps]
channel
$\quad$ ch[k|k:K•k ∈ ks] MSG

We leave channel messages. m:MSG, undefined.

6.1.3 Process Definitions 95

value
  system: W → **process**
  system(w) ≡
    ‖{comp_process(uid_Π(c))(c)|c:C•c ∈ w} ‖ ‖{atom_process(uid_Π(a),a)|a:A•a ∈ w}

  comp_process: π:Π → c:C→ **in**,**out** {ch(k)|k:K•k ∈ cm(π)} **process**
  comp_process(π)(c) ≡ [**assert:** π = uid_Π(c)]
   $\mathcal{M}_\mathcal{C}$(π)(c)(atr_ATR(c)) ‖
   ‖ {comp_process(uid_Π(c′))(c′)|c′:C•c′ ∈ obs_Ps(c)} ‖
   ‖ {atom_process(uid_Π(a))(a)|a:A•a ∈ obs_Ps(c)}

  $\mathcal{M}_\mathcal{C}$: π:Π → C → ATR → **in**,**out** {ch(k)|k:K•k ∈ cm(pi)} **process**
  $\mathcal{M}_\mathcal{C}$(π)(c)(c_attrs) ≡ $\mathcal{M}_\mathcal{C}$(c)($C\mathcal{F}$(c)(c_attrs)) **assert:** atr_ATR(c) ≡ c_attrs

  C$\mathcal{F}$: c:C → ATR → **in**,**out** {ch[em(i)]|i:KI•i ∈ cm(uid_Π(c))} ATR

ATR and atr_ATR are defined in Items 15.′ and 16.′ (Page 22). 96

  atom_process: a:A → **in**,**out** {ch[cm(k)]|:K•k ∈ cm(uid_Π(a))} **process**
  atom_process(a) ≡ $\mathcal{M}_\mathcal{A}$(a)(atr_ATR(a))

  $\mathcal{M}_\mathcal{A}$: a:A → ATR → **in**,**out** {ch[cm(k)]|k:K•k ∈ cm(uid_Π(a))} **process**
  $\mathcal{M}_\mathcal{A}$(a)(a_attrs) ≡ $\mathcal{M}_\mathcal{A}$(a)($A\mathcal{F}$(a)(a_attrs)) **assert:** atr_ATR(a) ≡ a_attrs

  A$\mathcal{F}$: a:A → ATR → **in**,**out** {ch[em(k)]|k:K • k ∈ cm(uid_Π(a))} ATR

97
The meaning processes $\mathcal{M}_\mathcal{C}$ and $\mathcal{M}_\mathcal{A}$ are generic. Their sôle purpose is to provide a never ending recursion. "In-between" they "make use" of Composite, respectively Atomic specific $\mathcal{F}$unctions here symbolised by $C\mathcal{F}$, respectively $A\mathcal{F}$.

Both $C\mathcal{F}$ and $A\mathcal{F}$ are expected to contain input/output clauses referencing the channels of their signatures; these clauses enable the sharing of attributes. We illustrate this "sharing" by the schematised function $\mathcal{F}$ standing for either $C\mathcal{F}$ or $A\mathcal{F}$. 98

value
  $\mathcal{F}$: p:(C|A) → ATR → **in**,**out** {ch[em(k)]|k:K • k ∈ cm(uid_Π(p))} ATR
  $\mathcal{F}$(p)(π,πs,props) ≡
     ⌈⌉ {**let** av = ch[em({π,j})] ? **in**
       ... ; [optional] ch[em({π,j})] ! in_reply(props)(av);
         (π,πs,in_update_ATR(props)(j,av)) **end** | {π,j}:K•{π,j} ∈ πs}
    ⌈⌉ ⌈⌉ { ... ; ch[em({π,j})] ! out_reply(props);
       (π,πs,out_update_ATR(props)(j)) | {π,j}:K•{π,j} ∈ πs}
     ⌈⌉ (π,πs,own_work(props))
  assert: π = uid_Π(p)

in_reply: Props → Π × VAL → VAL
in_update_ATR: Props → Π × VAL → Props
out_reply: Props → VAL
out_update_ATR: Props →Π → Props
own_work: Props → Props

We leave VAL undefined.

6.2 Discussion 99

6.2.1 General

A little more meaning has been added to the notions of parts and connections. The within and adjacent to relations between parts (composite and atomic) reflect a phenomenological world of geometry, and the connected relation between parts reflect both physical and conceptual world understandings: physical world in that, for example, radio waves cross geometric "boundaries", and conceptual world in that ontological classifications typically reflect lattice orderings where *overlaps* likewise cross geometric "boundaries".

6.2.2 Partial Evaluation 100

The composite_processes function "first" "functions" as a compiler. The 'compiler' translates an assembly structure into three process expressions: the $\mathcal{M}_\mathcal{C}(\mathsf{c})(\mathsf{c\_attrs})$ invocation, the parallel composition of composite processes, c', one for each composite sub-part of c, and the parallel composition of atomic processes, a, one for each atomic sub-part of c — with these three process expressions "being put in parallel". The recursion in composite_processes ends when a sub-...-composites consist of no sub-sub-...-composites. Then the compiling task ends and the many generated $\mathcal{M}_\mathcal{C}(\mathsf{c})(\mathsf{c\_attrs})$ and $\mathcal{M}_\mathcal{A}(\mathsf{a})(\mathsf{a\_attrs})$ process expressions are invoked.

7 Concluding Remarks 101

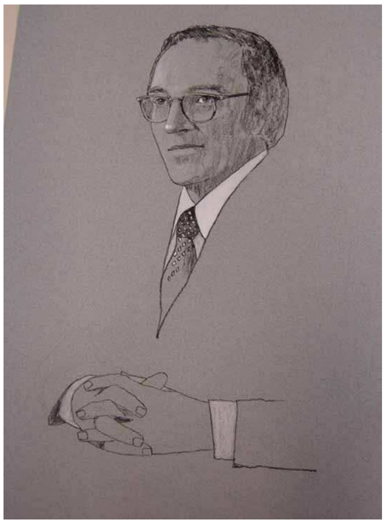
7.1 Relation to Other Work

The present contribution has been conceived in the following context.

My first awareness of the concept of 'mereology' was from listening to many presentations by **Douglas T. Ross** (1929–2007) at IFIP working group WG3.2 meetings over the years 1980–1999. In [15] Douglas T. Ross and John E. Ward reports on the 1958–1967 MIT project for *computer-aided design (CAD) for numerically controlled production*.[7] Pages 13–17 of [15] reflects on issues bordering to and behind the concerns of mereology. Ross' thinking is clearly seen in the following text: "... *our consideration of fundamentals begins not with design or problem-solving or programming or even mathematics, but with philosophy (in the old-fashioned meaning of the word) – we begin by establishing a "world-view". We have repeatedly emphasized that there is no way to bound or delimit the potential areas of application of our system, and that we must be prepared to cope with any conceivable problem. Whether the system will assist in any way in the solution of a given problem is quite another matter, ..., but in order to have a firm and uniform foundation, we must have a uniform philosophical basis upon which to approach any given problem. This "world-view" must provide a working framework and methodology in terms of which any aspect of our awareness of the world may be viewed. It must be capable of expressing the utmost in reality, giving expression to unending layers of ever-finer and more concrete detail, but at the same time abstract chimerical visions bordering on unreality must fall within the same scheme. "Above all, the world-view itself must be concrete and workable, for it will form the basis for all involvement of the computer in the problem-solving process, as well as establishing a*

[7] Doug is said to have coined the term and the abbreviation CAD [13].

viewpoint for approaching the unknown human component of the problem-solving team." Yes, indeed, the philosophical disciplines of ontology, epistemology and mereology, amongst others, ought be standard curricula items in the computer science and software engineering studies, or better: domain engineers cum software system designers ought be imbued by the wisdom of those disciplines as was Doug. *"... in the summer of 1960 we coined the word plex to serve as a generic term for these philosophical ruminations. "Plex" derives from the word plexus, "An interwoven combination of parts in a structure", (Webster). ... The purpose of a* **'modeling plex'** *is to represent completely and in its entirety a "thing", whether it is concrete or abstract, physical or conceptual. A 'modeling plex' is a trinity with three primary aspects, all of which must be present. If any one is missing a complete representation or modeling is impossible. The three aspects of plex are* **data, structure,** *and* **algorithm.** *... "* which *"... is concerned with the behavioral characteristics of the plex model– the interpretive rules for making meaningful the data and structural aspects of the plex, for assembling specific instances of the plex, and for interrelating the plex with other plexes and operators on plexes. Specification of the algorithmic aspect removes the ambiguity of meaning and interpretation of the data structure and provides a complete representation of the thing being modeled."* In the terminology of the current paper a plex is a part (whether composite or atomic), the data are the properties (of that part), the structure is the mereology (of that part) and the algorithm is the process (for that part). Thus Ross was, perhaps, a first instigator (around 1960) of object-orientedness. A first, "top of the iceberg" account of the mereology-ideas that Doug had then can be found in the much later (1976) three page note [14]. Doug not only 'invented' CAD but was also the father of AED (Algol Extended for Design), the Automatically Programmed Tool (APT) language, SADT (Structured Analysis and Design Technique) and helped develop SADT into the IDEF0 method for the Air Force's Integrated Computer-Aided Manufacturing (ICAM) program's IDEF suite of analysis and design methods. Douglas T. Ross went on for many years thereafter, to deepen and expand his ideas of relations between mereology and the programming language concept of type at the IFIP WG2.3 working group meetings. He did so in the, to some, enigmatic, but always fascinating style you find on Page 63 of [14].



Douglas T. Ross 1927–2007.
Courtesy MIT Museum

In [12] **Henry S. Leonard** and **Henry Nelson Goodman:** *A Calculus of Individuals and Its Uses* present the American Pragmatist version of Leśniewski's mereology. It is based on a single primitive: discreet, $\rceil$. The idea the calculus of individuals is, as in Leśniewski's mereology, to avoid having to deal with the empty sets while relying on explicit reference to classes (or parts).

[6] **R. Casati** and **A. Varzi:** *Parts and Places: the structures of spatial representation* has been the major source for this paper's understanding of mereology. Although our motivation was not the spatial or topological mereology, [16], and although the present paper does not utilize any of these concepts' axiomatisation in [6, 16] it is best to say that it has benefitted much from these publications.

Domain descriptions, besides mereological notions, also depend, in their successful form. on FCA: Formal Concept Analysis. Here a main inspiration has been drawn , since the mid 1990s from **B. Ganter** and **R. Wille's** *Formal Concept Analysis — Mathematical Foundations* [7]. *The approach takes as input a matrix specifying a set of objects and the properties thereof, called attributes, and finds both all the "natural" clusters of attributes and all the "natural" clusters of objects in the input data, where a "natural" object cluster is the set of all objects that share a common subset of attributes, and a "natural" property cluster is the set of all attributes shared by one of the natural object clusters. Natural property clusters correspond one-for-one with natural object clusters, and a concept is a pair containing both a natural property cluster and its corresponding natural object cluster. The family of these concepts obeys the mathematical axioms defining a lattice, a Galois connection).* Thus the choice of adjacent and embedded ('within') parts and their connections is determined after serious formal concept analysis. In [5] we present a 'concept analysis' approach to domain description, where the present paper presents the mereological approach.

The present paper is based on [3] of which it is an extensive revision and extension.

7.2 What Has Been Achieved ? 102

We have given a model-oriented specification of mereology. We have indicated that the model satisfies a widely known axiom system for mereology. We have suggested that (perhaps most) work on mereology amounts to syntactic studies. So we have suggested one of a large number of possible, schematic semantics of mereology. And we have shown that to every mereology there corresponds a set of communicating sequential process (CSP).

7.3 Future Work 103

We need to characterise, in a proper way, the class of CSP programs for which there corresponds a mereology. Are you game ?

One could also wish for an extensive editing and publication of Doug Ross' surviving notes.

7.4 Acknowledgements 104

I thank Dr. Claudio Calosi and Dr. Pierluigi Graziani, University of Urbino, Italy, for inviting this paper for a Springer Verlag *Synthese Library* volume on *Mereology and the Sciences*. I further thank Patricia M. Ross for permission to dedicate this paper to the memory of her husband of many years.

7.5 References

[1] D. Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling; Vol. 2: Specification of Systems and Languages; ol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.

[2] D. 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.

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

[4] D. Bjørner. The Role of Domain Engineering in Software Development. Why Current Requirements Engineering Seems Flawed! In *Perspectives of Systems Informatics*, volume 5947 of *Lecture Notes in Computer Science*, pages 2–34, Heidelberg, Wednesday, January 27, 2010. Springer.

[5] D. Bjørner and A. Eir. Compositionality: Ontology and Mereology of Domains. Some Clarifying Observations in the Context of Software Engineering in July 2008, eds. Martin Steffen, Dennis Dams and Ulrich Hannemann. In *Festschrift for Prof. Willem Paul de Roever Concurrency, Compositionality, and Correctness*, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59, Heidelberg, July 2010. Springer.

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

[7] B. Ganter and R. Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer-Verlag, January 1999. ISBN: 3540627715, 300 pages, Amazon price: US $ 44.95.

[8] C. W. George, P. Haff, K. Havelund, A. E. Haxthausen, R. Milne, C. B. Nielsen, S. Prehn, and K. R. Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

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

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

[11] C. Lejewski. A note on Leśniewksi's Axiom System for the Mereological Notion of Ingredient or Element. *Topoi*, 2(1):63–71, June, 1983.

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

[13] D. T. Ross. Computer-aided design. *Commun. ACM*, 4(5):41–63, 1961.

[14] D. 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.

[15] D. T. Ross and J. E. Ward. Investigations in computer-aided design for numerically controlled production. Final Technical Report ESL-FR-351, , May 1968. 1 December 1959 – 3 May 1967. Electronic Systems Laboratory Electrical Engineering Department, MIT, Cambridge, Massachusetts 02139.

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

[17] WWW. Domain Descriptions:

1. *A Container Line Industry Domain:* `http://www2.imm.dtu.dk/~db/container-paper.pdf`
2. *What is Logistics:* `http://www2.imm.dtu.dk/~db/logistics.pdf`
3. *The "Market": Consumers, Retailers, Wholesalers, Producers* `http://www2.imm.dtu.dk/~db/themarket.pdf`
4. *MITS: Models of IT Security. Security Rules & Regulations:* `http://www2.imm.dtu.dk/~db/it-security.pdf`
5. *A Domain Model of Oil Pipelines:* `http://www2.imm.dtu.dk/~db/pipeline.pdf`
6. *A Railway Systems Domain:* `http://www.railwaydomain.org/PDF/tb.pdf`
7. *Transport Systems:* `http://www2.imm.dtu.dk/~db/comet/comet1.pdf`
8. *The Tokyo Stock Exchange* `http://www2.imm.dtu.dk/~db/todai/tse-1.pdf` and `http://www2.imm.dtu.dk/~db/todai/tse-2.pdf`
9. *On Development of Web-based Software. A Divertimento of Ideas and Suggestions:* `http://www2.imm.dtu.dk/~db/wfdftp.pdf`

. R&D Experiments, Dines Bjørner, DTU Informatics, Technical University of Denmark, 2007–2010.