Chapter 12
A Rôle for Mereology in Domain Science and Engineering: To Every Mereology There Corresponds a λ–Expression

Dines Bjørner

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

12.1 Introduction

The term 'mereology' is accredited to the Polish mathematician, philosopher and logician Stansław Leśniewski (1886–1939). In this contribution we shall be concerned with only certain aspects of mereology, namely those that appear most immediately relevant to domain science (a relatively new part of current computer science). Our knowledge of 'mereology' has been through studying, amongst others, Casati and Varzi (1999) and Lejewski (1983).

12.1.1 Computing Science Mereology

"Mereology (from the Greek $\mu\epsilon\rho o\varsigma$ 'part') is the theory of part-hood relations: of the relations of part to whole and the relations of part to part within a whole".[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

---

[1] See Sect. 12.7.1.

[2] Achille Varzi: Mereology, http://plato.stanford.edu/entries/mereology/ 2009 and Casati and Varzi (1999)

D. Bjørner (✉)
Fredsvej 11, DK-2840 Holte, Denmark
e-mail: bjorner@gmail.com

Fig. 12.1 'Adjacent' and "embedded within" parts
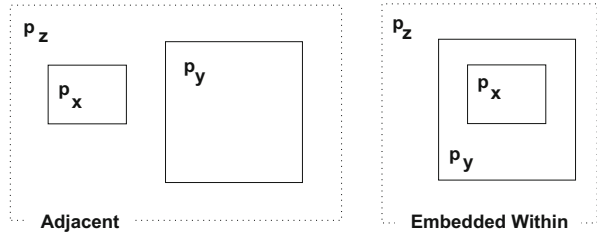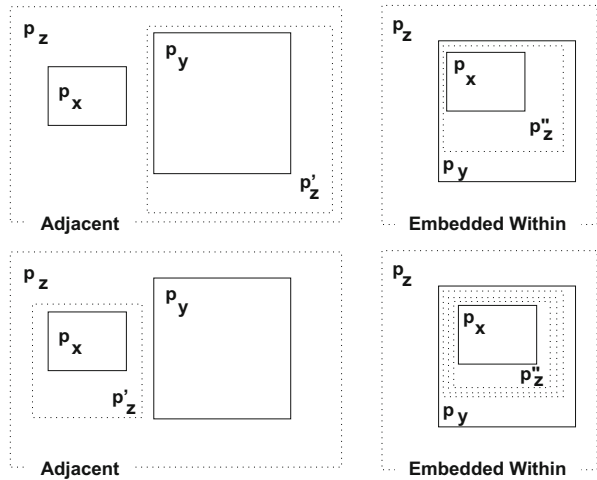

Fig. 12.2 'Adjacent' and "embedded within" parts

conceptually related. The relation: "being based", shall be made clear in this contribution.

Accordingly two parts, p_x and p_y, (of a same "whole") are either "adjacent", or are "embedded within" one another as loosely indicated in Fig. 12.1.

'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. 12.2 or one is "embedded within" the other—etc. as loosely indicated in Fig. 12.2.

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. We refer to Fig. 12.3.

Instead of depicting parts sharing properties as in the [L]eft side of Fig. 12.3, where dashed rounded edge rectangles stands for 'sharing', we shall (eventually) show parts sharing properties as in the [R]ight side of Fig. 12.3 where ●—● connections connect those parts.
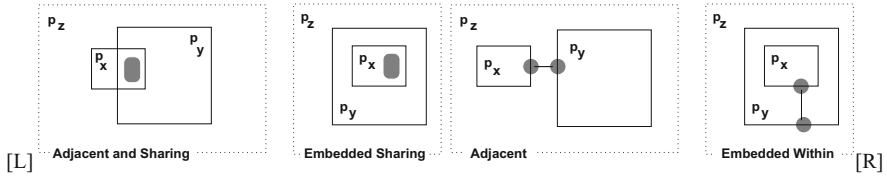
Fig. 12.3 Two models, [L,R], of parts that share properties

12.1.2 *From Domains via Requirements to Software*

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 development 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 (Bjørner 2008, 2010). Thus to properly *design* $\mathbb{S}$oftware we need to *understand* its or their $\mathbb{R}$equirements; and to properly *prescribe* $\mathbb{R}$equirements one must *understand* its $\mathbb{D}$omain. To *argue* correctness of $\mathbb{S}$oftware with respect to $\mathbb{R}$equirements one must usually *make assumptions* about the $\mathbb{D}$omain: $\mathbb{D}, \mathbb{S} \models \mathbb{R}$. Thus *description* of $\mathbb{D}$omains become an indispensable part of $\mathbb{S}$oftware development.

12.1.3 *Domains: Science and Engineering*

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.

12.1.4 *Contributions of This Contribution*

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 (Hoare 2004) program; and, related to (ii), (iii) suggesting complementing syntactic and semantic theories of mereology.

12.1.5   Structure of This Contribution

We briefly overview the structure of this contribution. First, in Sect. 12.2, we loosely characterise how we look at mereologies: *"what they are to us !"*. Then, in Sect. 12.3, 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. 12.4) focuses on the syntax of mereologies. Following that, in Sect. 12.4 we indicate how the model of Sect. 12.3 satisfies the axiom system of that section. In preparation for Sect. 12.6, Sect. 12.5 presents characterisations of attributes of parts, whether atomic or composite. Finally Sect. 12.6 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 12.7 concludes with some remarks on what we have achieved.

12.2   Our Concept of Mereology

12.2.1   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. 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 neighbouring street segments 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 example.

The mereological notion of subpart, that is: *contained within* can be illustrated by examples: *the intersections and street segments are subparts of the road net;*
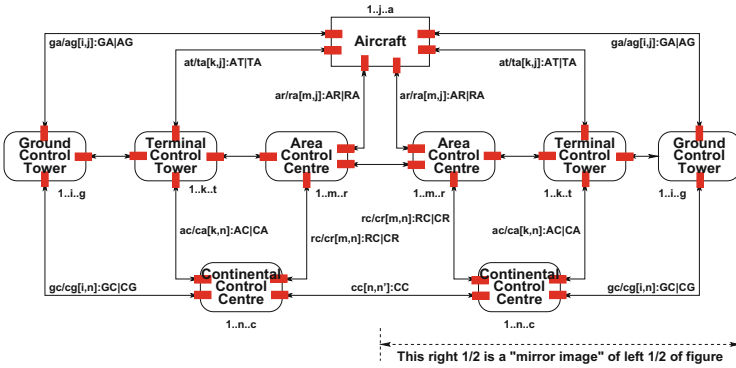
Fig. 12.4 A schematic air traffic system

vehicles are subparts of a platoon; and pipes, valves, pumps, forks and joins are subparts of pipelines. The mereological notion of adjacency can be illustrated by examples. We consider *the various controls of an air traffic system, cf. Fig. 12.4, as well as its aircrafts as adjacent within the air traffic system; the pipes, valves, forks, joins and pumps of a pipeline, cf. Fig. 12.9, as adjacent within the pipeline system; two or more banks of a banking system, cf. Fig. 12.6, 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 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".*

12.2.2   Six Examples

We shall, in Sect. 12.3, 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.

12.2.2.1   Air Traffic

Figure 12.4 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 shown. These index annotations are what makes the diagram of Fig. 12.4 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.

12.2.2.2 Buildings

Figure 12.5 shows a building plan—as a composite part.

The building consists of two buildings, A and H. The buildings A and H are neighbours, but shares a common wall. Building A has rooms B, C, D and E, Building H has rooms I, 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.

12.2.2.3 Financial Service Industry

Figure 12.6 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

---

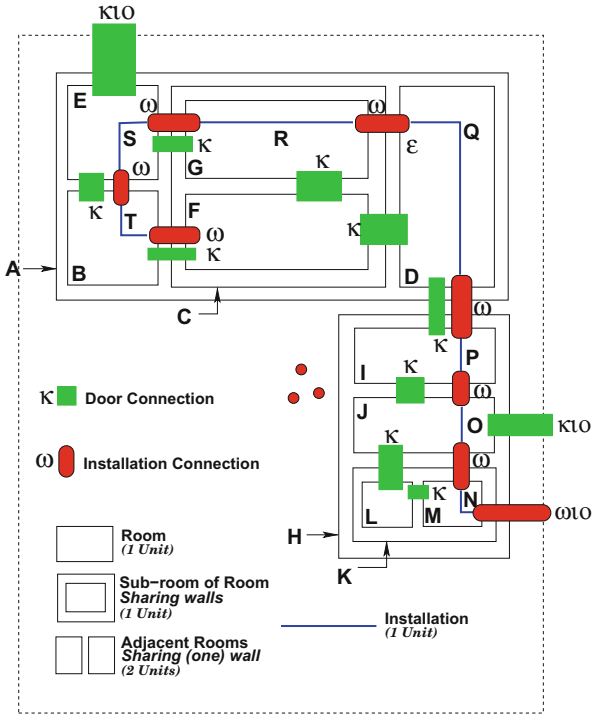[3]There are 38 such rectangles in Fig. 12.4.

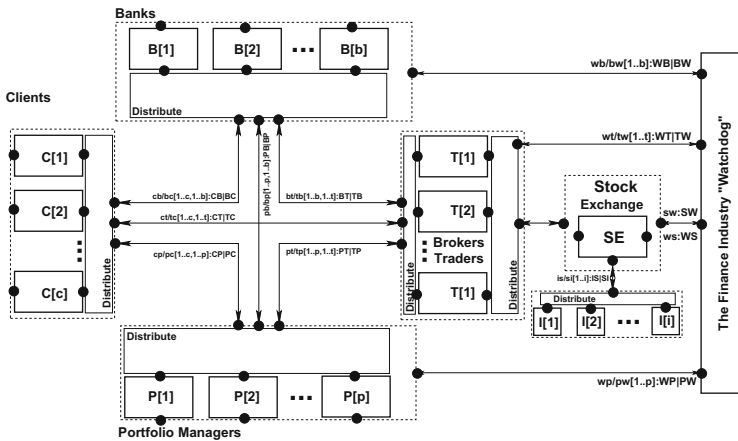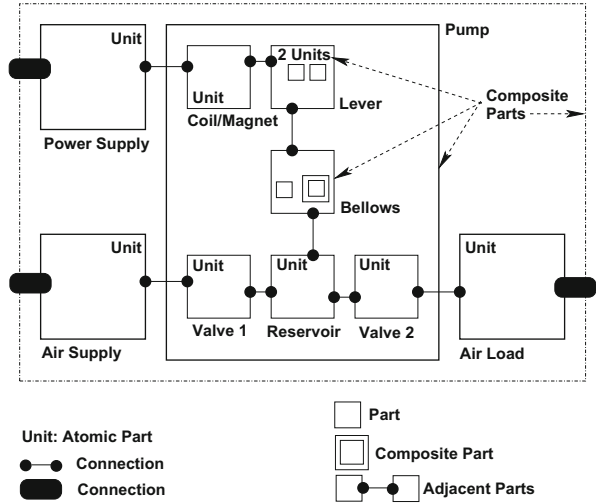**Fig. 12.5** A building plan with installation

Fig. 12.6 A financial service industry

Fig. 12.7 An air pump, i.e., a physical mechanical system

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, •.

12.2.2.4 Machine Assemblies

Figure 12.7 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) external connectors. Etcetera. Some connections afford "transmission" of electrical power. Other connections convey torque. Two connections convey input air, respectively output air.

12.2.2.5 Oil Industry

Figure 12.8 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.
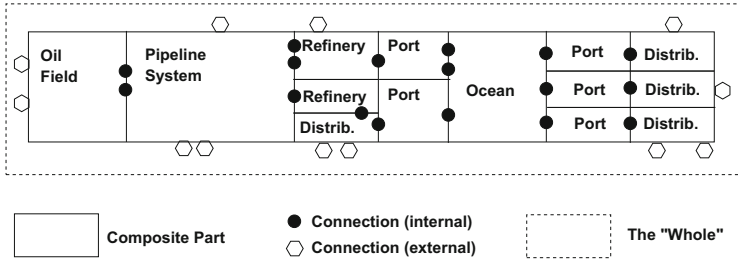
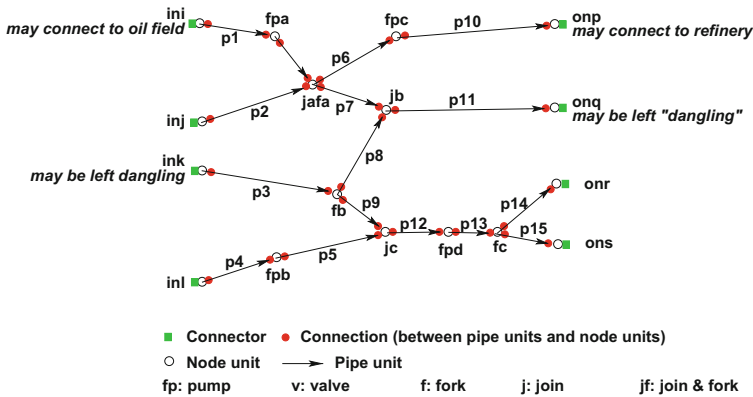**Fig. 12.8** A schematic of an oil industry

Fig. 12.9 A pipeline system

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.

Figure 12.9 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 ini–$in\ell$), four (4) flow pump units (shown as small circles, ∘, and labelled fpa–fpd), 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 $jafa$), and four (4) output node units (shown as small circles, ∘, and labelled onp–ons).

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.

Fig. 12.10 Four example rail units
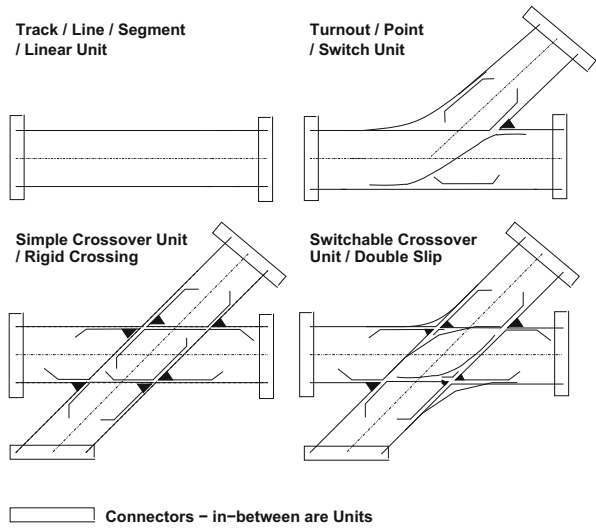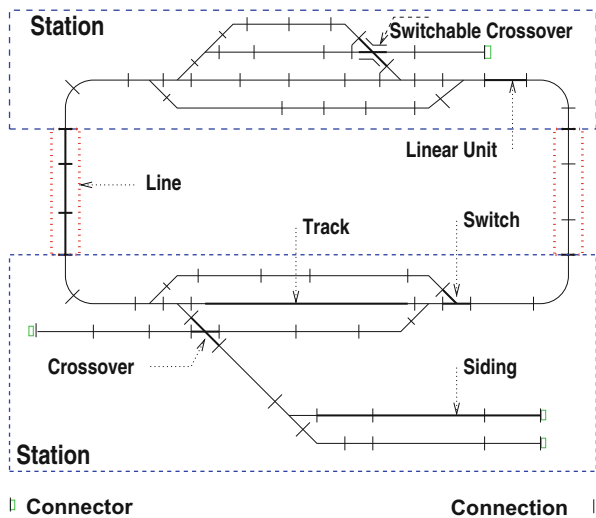

Connectors − in−between are Units

Fig. 12.11 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. 12.10. There are 66 connections and four "dangling" connectors

12.2.2.6 Railway Nets

Figure 12.10 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. 12.11 into proper rail nets.

Figure 12.11 diagrams an example of a proper rail net. It is assembled from the kind of units shown in Fig. 12.10. In Fig. 12.11 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. 12.10 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.

12.2.2.7   Discussion

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. WWW (2007–2010) gives URLs to a number of domain models illustrating a great variety of mereologies.

12.3   An Abstract, Syntactic Model of Mereologies

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.

12.3.1   Parts and Subparts

Figure 12.12 illustrates composite and atomic parts. The *slanted sans serif* uppercase identifiers of Fig. 12.12 *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".

12.3.1.1   The Model

The formal models of this contribution are expressed in the RAISE Specification Language, RSL (Bjørner 2006; George et al. 1992, 1995).

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.
4. Atomic parts cannot be confused with composite parts.

Fig. 12.12 Atomic and
composite parts
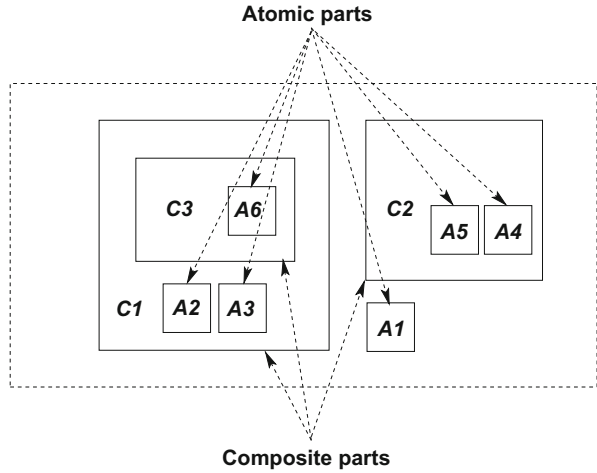
5. From a composite part one can observe one or more parts.

type
1. W = P-**set**
2. P = A | C
value
3. is_A: P → **Bool**, is_C: P → **Bool**
axiom
4. ∀ a:A,c:C•a≠c, i.e., A∩C={‖} ∧ is_A(a)≡∼is_C(a)∧is_C(c)≡∼is_A(c)
value
5. obs_Ps: C → P-**set**   **axiom** ∀ c:C • obs_Ps(c)≠{}

The type expression {‖} notes the empty type.
   Figure 12.12 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*}.

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**.

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**

$ps \cup \cup \{$ set-of-sets $\}$ expresses that the set ps is union'ed (the first $\cup$, left-to-right) with the distributed union (the second $\cup$, left-to-right) of the set of sets.

12.3.2 'Within' and 'Adjacency' Relations

12.3.2.1 '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: $P \times P \overset{\sim}{\to}$ **Bool**
7. imm_within$(p,p') \equiv$
7a.   is_C(p')
7b. $\land\ p \in$ obs_Ps(p')

12.3.2.2 'Transitive Within'

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: $P \times P \overset{\sim}{\to}$ **Bool**
8. within$(p,p') \equiv$
8a.   imm_within(p,p')
8b. $\lor\ \exists\ p'':C \bullet p'' \in$ obs_Ps$(p') \land$ within(p,p'')

12.3.2.3 'Adjacency'

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: $P \times P \to C \overset{\sim}{\to}$ **Bool**,
9. imm_adjacent$(p,p')(c) \equiv p \neq p' \land \{p,p'\} \subseteq$ obs_Ps(c)

12.3.2.4 Transitive 'Adjacency'

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))

12.3.3 Unique Identifications

Each physical part can always 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: $P \rightarrow \Pi$
axiom
13. ∀ p,p′:P • p≠p′ ⇒ uid_Π(p)≠uid_Π(p′)

Figure 12.13 illustrates the unique identifications of composite and atomic parts.
We exemplify the observer function obs_Π in the expressions below and on Fig. 12.13 on the facing page:

Fig. 12.13 ai_j: atomic part identifiers, ci_k: composite part identifiers
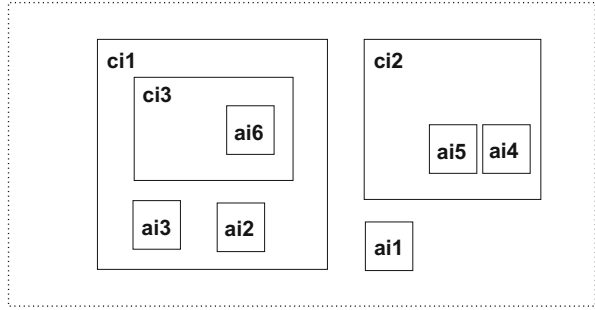
- obs_Π(*C1*) = *ci1*, obs_Π(*C2*) = *ci2*, etcetera; and
- obs_Π(*A1*) = *ai1*, obs_Π(*A2*) = *ai2*, etcetera.

Please note that also this example is meta-linguistic.

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

12.3.4   Attributes

In Sect. 12.5 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 $\in$.

type
15.   ATR
value
16.   atr_ATRs: P $\rightarrow$ ATR-**set**
17.   share: P$\times$P $\rightarrow$ **Bool**
17.   share(p,p$'$) $\equiv$ p$\neq$p$'$$\wedge$$\exists$ atr:ATR•atr $\in$ atr_ATRs(p)$\wedge$atr $\in$ atr_ATRs(p$'$)
18. $\in$: ATR $\times$ ATR-**set** $\rightarrow$ **Bool**

Fig. 12.14 Connectors
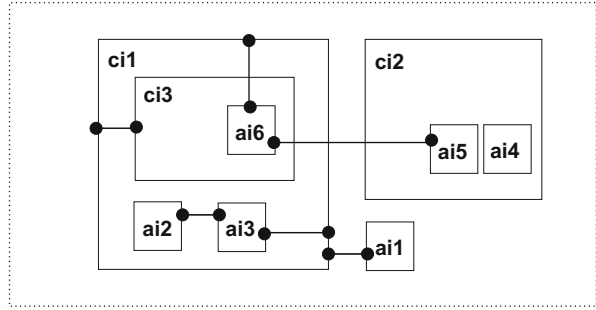
12.3.5 Connections

In order to illustrate other than the within and adjacency part relations we introduce the notions of connectors and, hence, connections. Figure 12.14 illustrates connections between parts. A connector is, visually, a ●—● line that connects two distinct part boxes.

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
    - $\{ai_1, ci_1\}$.

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.   xtr_Ks: P → K-**set**
21.   xtr_Ks(p) ≡ {{uid_Π(p),π}|π:Π•π ∈ mereo_Πs(p)}

12.3.5.1   Connector Wellformedness

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. wf_Ks: C → **Bool**
22. wf_Ks(c) ≡
23.   **let** ks = xtr_Ks(c), πs = mereo_Πs(c) **in**
24.   ∀ {π',π''}:Π-**set** • {π',π''}⊆ks ⇒
24.   ∃ p′,p″:P • {π',π''}={uid_Π(p′),uid_Π(p″)} **end**

12.3.5.2 Connector and Attribute Sharing Axioms

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. ∀ w:W•
25.   **let** ps = xtr_Ps(w), ks = xtr_Ks(w) **in**
25a.   ∀ p,p′:P • p≠p′ ∧ {p,p′}⊆ps ∧ share(p,p′) ⇒
25a.   {uid_Π(p),uid_Π(p′)} ∈ ks ∧
25b.   ∀ {uid,uid′} ∈ ks ⇒
25b.   ∃ p,p′:P • {p,p′}⊆ps ∧ {uid,uid′}={uid_Π(p),uid_Π(p′)} ⇒
25b.   share(p,p′) **end**
value
26. xtr_Ks: W → K-**set**
26. xtr_Ks(w) ≡ ∪{xtr_Ks(p)|p:P•p ∈ obs_Ps(p)}

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

12.3.5.3 Sharing

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

27. sharing: P × P → **Bool**
28. sharing(p,p′) ≡ p≠p′∧share(p,p′)

12.3.6   Uniqueness of Parts

There is one property of the model of wholes: W, Item 1, and hence the model of composite and atomic parts and their unique identifiers "spun off" from W (Item 2 [Page 333] to Item 25b [Page 339]). 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.

12.3.6.1   Uniqueness of Embedded and Adjacent Parts

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: <u>u</u>nique <u>i</u>mmediate <u>id</u>entifier<u>s</u>).

value
29. uiids: C → **Bool**
29. uiids(c) ≡
29.   ∀ p,p':P•p≠p'∧{p,p'}⊆obs_Ps(c)⇒**card**{uidΠ(p),uidΠ(p'),uidΠ(c)}=3

30. We must now specify that that uniqueness is "propagated" to parts that are proper parts of parts of a composite part (uids: <u>u</u>nique <u>id</u>entifier<u>s</u>).

30. uids: C → **Bool**
30. uids(c) ≡
30.   ∀ c':C•c' ∈ obs_Ps(c) ⇒ uiids(c')
30.   ∧ **let** ps'=xtr_Ps(c'),ps''=xtr_Ps(c'') **in**
30.   ∀ c'':C•c'' ∈ ps'⇒uids(c'')
30.   ∧ ∀ p',p'':P•p' ∈ ps'∧p'' ∈ ps''⇒uid_Π(p')≠uid_Π(p'') **end**

12.4   An Axiom System

Classical axiom systems for mereology focus on just one sort of "things", namely $\mathscr{P}$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.

12.4.1   Parts and Attributes

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

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

$\mathscr{A}$ttributes are associated with $\mathscr{P}$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 $\mathscr{P}$arts and $\mathscr{A}$ttributes.

- $\in: \mathscr{A} \times \mathscr{P} \to$ **Bool**.

12.4.2   The Axioms

The axiom system to be developed in this section is a variant of that in Casati and Varzi (1999). We introduce the following relations between parts[5]:

$$
\begin{aligned}
\text{part\_of:} \quad & \mathbb{P} : \mathscr{P} \times \mathscr{P} \to \textbf{Bool} \\
\text{proper\_part\_of:} \quad & \mathbb{PP} : \mathscr{P} \times \mathscr{P} \to \textbf{Bool} \\
\text{overlap:} \quad & \mathbb{O} : \mathscr{P} \times \mathscr{P} \to \textbf{Bool} \\
\text{underlap:} \quad & \mathbb{U} : \mathscr{P} \times \mathscr{P} \to \textbf{Bool} \\
\text{over\_crossing:} \quad & \mathbb{OX} : \mathscr{P} \times \mathscr{P} \to \textbf{Bool} \\
\text{under\_crossing:} \quad & \mathbb{UX} : \mathscr{P} \times \mathscr{P} \to \textbf{Bool} \\
\text{proper\_overlap:} \quad & \mathbb{PO} : \mathscr{P} \times \mathscr{P} \to \textbf{Bool} \\
\text{proper\_underlap:} \quad & \mathbb{PU} : \mathscr{P} \times \mathscr{P} \to \textbf{Bool}
\end{aligned}
$$

Let $\mathbb{P}$ denote part-hood; p_x is part of p_y, is then expressed as $\mathbb{P}(p_x, p_y)$. Equation (12.1) Part p_x is part of itself (reflexivity). Equation (12.2) 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). Equation (12.3) 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).

$$\forall p_x : \mathscr{P} \bullet \mathbb{P}(p_x, p_x) \tag{12.1}$$

$$\forall p_x, p_y : \mathscr{P} \bullet (\mathbb{P}(p_x, p_y) \wedge \mathbb{P}(p_y, p_x)) \Rightarrow p_x = p_y \tag{12.2}$$

$$\forall p_x, p_y, p_z : \mathscr{P} \bullet (\mathbb{P}(p_x, p_y) \wedge \mathbb{P}(p_y, p_z)) \Rightarrow \mathbb{P}(p_z, p_z) \tag{12.3}$$

---

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

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

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.

$$\mathbb{PP}(p_x, p_y) \triangleq \mathbb{P}(p_x, p_y) \wedge \neg\mathbb{P}(p_y, p_x) \tag{12.4}$$

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.

$$\mathbb{O}(p_x, p_y) \triangleq \exists a : \mathscr{A} \bullet a \in p_x \wedge a \in p_y \tag{12.5}$$

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.

$$\mathbb{U}(p_x, p_y) \triangleq \exists p_z : \mathscr{P} \bullet \mathbb{P}(p_x, p_z) \wedge \mathbb{P}(p_y, p_z) \tag{12.6}$$

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.

$$\mathbb{OX}(p_x, p_y) \triangleq \mathbb{O}(p_x, p_y) \wedge \neg\mathbb{P}(p_x, p_y) \tag{12.7}$$

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.

$$\mathbb{UX}(p_x, p_y) \triangleq \mathbb{U}(p_x, p_z) \wedge \neg\mathbb{P}(p_y, p_x) \tag{12.8}$$

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.

$$\mathbb{PO}(p_x, p_y) \triangleq \mathbb{OX}(p_x, p_y) \wedge \mathbb{OX}(p_y, p_x) \tag{12.9}$$

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.

$$\mathbb{PU}(p_x, p_y) \triangleq \mathbb{UX}(p_x, p_y) \wedge \mathbb{UX}(p_y, p_x) \tag{12.10}$$

12.4.3 Satisfaction

We shall sketch a proof that the *model* of the previous section, Sect. 12.3, *satisfies*—is a model for—the *axioms* of this section. To that end we first define the notions of *interpretation, 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.

12.4.3.1 A Proof Sketch

We assign

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

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

12.5 An Analysis of Properties of Parts

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. 12.6 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]

12.5.1 *Mereological Properties*

12.5.1.1 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 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.

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** $\forall$ l:L•**card** mereo_HIs(l)=2
     mereo_LIs: H→LI-**set**
axiom
   $\forall$ n:N•
      let hs=obs_Hs(n),ls=obs_Ls(n) **in**
      $\forall$ h:H•h $\in$ hs$\Rightarrow$$\forall$ li:LI•li $\in$ mereo_LIs(h)$\Rightarrow$$\exists$ l:L•uid_LI(l)=li
      $\wedge$ $\forall$ l:L•l $\in$ ls$\Rightarrow$$\exists$ h,h':H•{h,h'}$\subseteq$hs$\wedge$mereo_HIs(l)={uid_HI(h),uid_HI(h')}
      end   •

12.5.1.2 Unique Identifier and Mereology Types

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
        $\Pi = \Pi1 | \Pi2 | ... | \Pi$n
    value
        uid_Πj: Pj $\rightarrow$ Πj for 1$\leq$j$\leq$n
        mereo_Πs: P $\rightarrow$ Π-**set**

Example: Our example relates to the abstract model of Sect. 12.3.

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 $0 \leq j \leq n$.

type
37. Π
value
37. uid_Π: P $\rightarrow \Pi$
38. mereo_Πs: P $\rightarrow \Pi$-**set**
axiom
38. $\forall$ p:P•uid_Π(p) $\notin$ mereo_Πs(p)
value
39. xtr_Ks: P $\rightarrow$ K-**set**
39. xtr_Ks(p) $\equiv$
39.   **let** $(\pi, \pi s) = (\text{uid}\_\Pi, \text{mereo}\_\Pi s)(p)$ **in**
39. $\{\{\pi', \pi''\} | \pi', \pi'' : \Pi • \pi' = \pi \wedge \pi'' \in \pi s\}$ **end**

                                                                                    •

12.5.2 Properties

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 his claim ("their final meaning") in Sect. 12.6.) 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 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.   •

40. Parts are characterised (also) by a set of one or more distinctly named and not necessarily distinctly typed property values.

---

[6] For example, a simple 'carrefour', or a (circular) roundabout, or a free-way interchange, or 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.

   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.

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

We trust that the reader can decipher this example.   •

12.5.3   Attributes

There are (thus) three kinds of part attributes:

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

We refer to Sect. 12.3.4, and to Items 15–16.

type
15.$'$ ATR = $\Pi \times \Pi$**-set** $\times$ Props
value
16.$'$ atr_ATR: P $\rightarrow$ ATR
axiom
      $\forall$ p:P • **let** $(\pi,\pi s,props)$ = atr_ATR(p) **in** $\pi \notin \pi s$ **end**

In preparation for redefining the share function of Item 17 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 $\xrightarrow{m}$ PropVAL_or_Π
41. PropVAL_or_Π :: mk_Simp:PropVAL | mk_Π:Π

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.

value
42. get_VAL: P $\times$ PropName $\rightarrow$ W $\rightarrow$ PropVAL
42. get_VAL(p,pn)(w) $\equiv$
44.   **let** pv = (obs_Props(p))(pn) **in**
42.   **case** pv **of**
42.   mk_Simp(v) $\rightarrow$ v,
42a.   mk_$\Pi(\pi) \rightarrow$
42a.   **let** p':P•p' $\in$ xtr_Ps(w)$\wedge$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.

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. $\pi \notin \pi$s ∧
43b.   ∀ π':Π • $\pi' \in$ **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**

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.

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. $\pi \in \pi$s′ ∨ $\pi' \in \pi$s ∨
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.

12.5.4 Discussion

We have now witnessed four kinds of observer function:

- The 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 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 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. 12.3) we have not modelled distinctions between static and dynamic properties. You may think, instead of such a model, that an **always** temporal operator, □, being applied to appropriate predicates.

12.6   A Semantic CSP Model of Mereology

The model of Sect. 12.3 can be said to be an abstract model-oriented definition of the syntax of mereology. Similarly the axiom system of Sect. 12.4 can be said to be an abstract property-oriented definition of the syntax of mereology. With the analysis of attributes of parts, Sect. 12.5, we have begun a semantic analysis of mereology. We now bring that semantic analysis a step further.

12.6.1 A Semantic Model of a Class of Mereologies

We show that to every mereology there corresponds a program of cooperating sequential processes CSP (Hoare 2004).

Some of the attributes of parts may be static, i.e., constants, others may be dynamic, i.e., variable. The latter form the state of parts. Actions change part states. Processes, P, Q, are sets of sequences of actions and sets of processes. Processes may communicate values (of type M) and then do so via channels, ch. Schematic process P and Q definitions

type
[1]   A, B, M
    channel
[2]   ch:M
    value
[3]   P: A → **out** ch **process**,   Q: B → **in** ch **process**
[4]   P(a) ≡ ... ;   Q(b) ≡ ... ;
[5]   ch!f(a);   **let** v = ch? **in**
[6]   ... ;   ... ;
[7]   P(g(a))   Q(h(v,b)) **end**

[1] A, B and M are sets of values. [3] P and Q are names of processes. [4] Process
P initially accepts arguments a of type A, may offer outputs, ch!f(a), of type M
on channel ch and otherwise continues "infinitely" [7] (hence type **process** in line
[3]). Similarly Q initially accepts arguments of type B, may accept input, ch? (of
type M) also on channel ch and otherwise continues "infinitely" [7]. [5] The output
offering, ch!f(a), of process P may be accepted, ch?, by Q. The "..." accounts for
our saying "may". We leave the d and h functions undefined.

12.6.1.1   Parts ≃ Processes

The model of mereology presented in Sect. 12.3 (Pages 333–340) 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.

12.6.1.2   Connectors ≃ Channels

The CSP channels are indexed by the two-set (hence distinct) part identifier
connectors. From a whole we can extract (xtr_Ks, Item 26) all connectors. They
become indexes into an array of 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** = ∪{xtr_Ps(c)|c:C•c ∈ w} ∪ {a|a:A•a ∈ w}
    ks:K-**set** = xtr_Ks(w)
type
    K = Π-**set axiom** ∀ k:K•**card** k=2
    ChMap = Π $\overrightarrow{m}$ K-**set**
value
    cm:ChMap = [uid_Π(p)↦xtr_Ks(p)|p:P•p ∈ ps]

channel

　　ch[k|k:K•k ∈ ks] MSG

We leave channel messages. m:MSG, undefined.

12.6.1.3   Process Definitions

The whole, w:W, where W=P-set, is semantically seen as the distributed parallel composition of part processes one for each part, p_i in w where w = $\{p_1,p_2,\ldots,p_m\}$.

value

　　system: W → **process**
　　system(w) ≡ ‖ {part_process(uid_Π(p))(p)|p:P•p ∈ w}

A part process is either a composite (part) process or an atomic (part) process.

value

　　part_process: Π → P → **process**
　　part_process(π)(p) ≡ **assert:** π = uid_Π(p)
　　　　is_C(p) → composite_process(π)(p),
　　　　is_A(p) → atomic_process(π)(p)

A composite process, c, is the parallel composition of the core composite process, $\mathcal{M}_\mathcal{C}$, with the distributed parallel composition of part processes, one for each part observed from c.

value

　　composite_process: π:Π → c:C → **in,out** {ch(k)|k:K•k ∈ cm(π)} **process**
　　composite_process(π)(c) ≡ **assert:** π = uid_Π(c)
　　　　$\mathcal{M}_\mathcal{C}$(π)(c)(atr_ATR(c)) ‖
　　‖ {part_process(uid_Π(p))(p)|p:P•ps ∈ obs_Ps(p)}

ATR and atr_ATR are defined in Items 15.′ and 16.′ (Page 347).

　　The core composite process, $\mathcal{M}_\mathcal{C}$ (of a composite part c), is an in[de]finite cyclic process which evolves around the attributes, atr_ATR(c), of c. $\mathcal{M}_\mathcal{C}$ is based on a postulated, i.e., an undefined attribute update action C$\mathcal{F}$.

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

C$\mathcal{F}$ potentially communicates with all those part processes (of the whole, w) with which c, the part on which $\mathcal{M}_\mathcal{C}$(π)(c)(atr_ATR(c)) is based, shares attributes, that is, has connectors.

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

Atomic processes are just that: They evolve sôlely around a core atomic process $\mathscr{M}_{\mathscr{A}}$(a)(atr_ATR(a)).

atomic_process: π:Π → A → **in,out** {ch[cm(k)]|:K•k ∈ cm(π)} **process**
atomic_process(π)(a) ≡ $\mathscr{M}_{\mathscr{A}}$(a)(atr_ATR(a)) **assert:** π = uid_Π(a)

The core atomic process, $\mathscr{M}_{\mathscr{A}}$($\pi$)(a)(atr_ATR(a)), is based on a postulated, i.e., an undefined attribute update action A$\mathscr{F}$.

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

A$\mathscr{F}$ potentially communicates with all those part processes (of the whole, w) with which a, the part on which $\mathscr{M}_{\mathscr{A}}$($\pi$)(a)(atr_ATR(a))is based, shares attributes, that is, has connectors.

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

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

Both $C\mathscr{F}$ and $A\mathscr{F}$ 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 $\mathscr{F}$ standing for either $C\mathscr{F}$ or $A\mathscr{F}$.

The $\mathscr{F}$ action non-deterministically internal choice chooses between either [1,2,3] accepting input from another part process, then optionally offering a reply to that other process, and finally delivering an updated state; or [4,5] offering an output to another part process, and then delivering an updated state; or [6] doing own work resulting in an updated state.

value
     $\mathscr{F}$: p:P → ATR → **in,out** {ch[em(k)]|k:K • k ∈ cm(uid_Π(p))} ATR
     $\mathscr{F}$(p)(π,πs,props) ≡ **assert:** uid_Π(p)=π
[1]   [] {**let** av = ch[em({π,j})] ? **in**
[2]   ... ; [optional] ch[em({π,j})] ! in_reply(props)(av);
[3]   (π,πs,in_update_ATR(props)(j,av)) **end** | {π,j}:K•{π,j} ∈ πs}
[4]   ⊓ [] { ... ; ch[em({π,j})] ! out_reply(props);
[5]   (π,πs,out_update_ATR(props)(j)) | {π,j}:K•{π,j} ∈ πs}
[6]   ⊓   (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.

12.6.2 Discussion

Parts, subparts and their relations, that is, mereology, reflect *syntactic properties* of wholes. The interpretations of parts as processes and mereology as channels reflect *semantic properties* of wholes. What we have shown in this section is that to every mereology there corresponds a *normal form* CSP "program schema".

12.7 Concluding Remarks

A first basic idea of this paper has been to to take axiom systems of mereology and render them mathematical in the sense of showing that a mathematical model satisfies the axiom system A second basic idea of this paper has then been to extend this model-oriented treatment to not just covering syntactic aspects of mereology but also to cover, albeit schematically, normative, schematic models of semantic aspects of mereology.

12.7.1 Relation to Other Work

The present contribution has been conceived in the following model-oriented 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 Douglas T. Ross and John E. Ward (1968) reports on the 1958–1967 MIT project for *computer-aided design (CAD) for numerically controlled production*.[7] Pages 13–17 of Ross and Ward (1968) 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*

---

[7]Doug is said to have coined the term and the abbreviation CAD (Ross 1961).

Douglas T. Ross 1927–2007.
Courtesy MIT Museum

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 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 (Ross 1976). 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 Ross (1976).

In Henry S. Leonard and Henry Nelson Goodman (1940): *A Calculus of Individuals and Its Uses* present the American Pragmatist version of Leśniewski's mereology. It is based on a single primitive: *discrete.* 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). The treatment of Leonard and Goodman (1940) is axiomatic.

R. Casati and A. Varzi (1999): *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, Smith (1996), and although the present paper does not utilize any of these concepts' axiomatisation in Casati and Varzi (1999) and Smith (1996) it is best to say that it has benefitted much from these publications. The treatments of these papers are axiomatic.

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* (Ganter and Wille 1999). *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 Bjørner and Eir (2008) we present a 'concept analysis' approach to domain description, where the present paper presents the mereological approach.

   The present paper is based on Bjørner (2009) of which it is an extensive revision and extension.

12.7.2   What Has Been Achieved?

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 `CSP` processes.

12.7.3   Future Work

There are four kinds of 'future works': (i) studies that give us further insight into the syntactic mereology operators: `overlap`, `underlap`, `over-crossing`, `under-crossing`, `proper overlap` and `proper underlap`; (ii) studies that explore further semantic models of mereology, we, for example, need to characterise the class of `CSP` programs for which there corresponds a mereology; (iii) refinements of the normative, schematics `CSP` (Sect. 12.6) models of mereology; and (iv) an extensive editing and publication of Doug Ross' surviving notes.

References

Bjørner, D. (2006). *Software engineering, Vol. 1: Abstraction and modelling; Vol. 2: Specification of systems and languages; Vol. 3: Domains, requirements and software design* (Texts in theoretical computer science, the EATCS series). Berlin/New York: Springer.

Bjørner, D. (2008). From domains to requirements. In P. Degano, Rocco De Nicola, & J. Meseguer (Eds.), *Montanari festschrift* (Lecture notes in computer science, pp. 1–30). Heidelberg: Springer.

Bjørner, D. (2009). On mereologies in computing science. In B. Roscoe (Ed.), *Festschrift for Tony Hoare* (History of computing london). UK: Springer.

Bjørner, D. (2010). 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, pp. 2–34). Heidelberg: Springer.

Bjørner, D., & Eir, A. (2010). Compositionality: Ontology and mereology of domains. Some clarifying observations in the context of software engineering in July 2008. In M. Steffen, D. Dams, & U. Hannemann (Eds.), *Festschrift for Prof. Willem Paul de Roever concurrency, compositionality, and correctness* (Volume 5930 of lecture notes in computer science, pp. 22–59). Heidelberg: Springer.

Casati, R., & Varzi, A. (1999). *Parts and places: The structures of spatial representation*. Cambridge: MIT.

Ganter, B., & Wille, R. (1999). *Formal concept analysis—mathematical foundations*. Springer. ISBN:3540627715, 300 pages, Amazon price: US $44.95.

George, C. W., Haff, P., Havelund, K., Haxthausen, A. E., Milne, R., Nielsen, C. B., Prehn, S., & Wagner, K. R. (1992). *The RAISE specification language* (The BCS practitioner series). Hemel Hampstead: Prentice-Hall.

George, C. W., Haxthausen, A. E., Hughes, S., Milne, R., Prehn, S., & Pedersen, J. S. (1995). *The RAISE development method* (The BCS practitioner series). Hemel Hampstead: Prentice-Hall.

Hoare, C. (1985). *Communicating sequential processes* (C.A.R. Hoare series in computer science). Prentice-Hall International. Published electronically: http://www.\discretionary-using\discretionary-csp.\discretionary-com/\discretionary-csp\discretionary-book.pdf (2004).

Lejewski, C. (1983). A note on Leśniewksi's axiom system for the ereological notion of ingredient or element. *Topoi, 2*(1), 63–71.

Leonard, H. S., & Goodman, N. (1940). The calculus of individuals and its uses. *Journal of Symbolic Logic, 5*, 45–44.

Ross, D. T. (1961). Computer-aided design. *Communications of the ACM, 4*(5), 41–63.

Ross, D. T. (1976). Toward foundations for the understanding of type. In *Proceedings of the 1976 conference on data: Abstraction, definition and structure*, Salt Lake City (pp. 63–65). New York: ACM.

Ross, D. T., & Ward, J. E. 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.

Smith, B. (1996). Mereotopology: A theory of parts and boundaries. *Data and Knowledge Engineering, 20*, 287–303.

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://euler.fd.cvut.cz/railwaydomain/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 Compute, Technical University of Denmark, 2007–2010.