On Mereologies in Computing Science

Dines Bjørner

In this paper we solve the following problems:

- we give a formal model of a large class of mereologies, with simple entities modelled as parts and their relations by connectors;
- we show that that class applies to a wide variety of societal infrastructure component domains;
- we show that there is a class of `CSP` channel and process structures that correspond to the class of mereologies where mereology parts become `CSP` processes and connectors become channels; and where simple entity attributes become process states.

We have yet to prove to what extent the models satisfy the axiom systems for mereologies of, for example, [13] and a calculus of individuals [14]. Mereology is the study, knowledge and practice of part-hood relations: of the relations of part to whole and the relations of part to part within a whole. By parts we shall here understand simple entities — of the kind illustrated in this paper.

Manifest simple entities of domains are either continuous (fluid, gaseous) or discrete (solid, fixed), and if the latter, then either atomic or composite. It is how the sub-entities of a composite entity are "put together" that "makes up" a mereology of that composite entity — at least such as we shall study the mereology concept. In this paper we shall study some ways of modelling the mereology of composite entities. One way of modelling mereologies is using sorts, observer functions and axioms (McCarthy style), another is using CSP.

Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

IFIP WG2.3: A Laudatio and a Memory

This paper is in honour of Sir Tony Hoare. And the paper is in memory of Douglas Taylor Ross (1929–2007). The latter speculated quite a lot about mereologies at many IFIP WG 2.3 meetings; not quite all members and observers understood everything; certainly not I. But I somehow knew it was a relevant issue. I think I now understand what Doug was saying. Here then, in this paper, is my interpretation of Doug's discourses. The former, today's celebrant, has given us many deep, yet simple, hence elegant, concepts. CSP is one of them. Therefore CSP will be applied, at the end of the paper, to express mereologies. IFIP WG 2.3 meetings in my days certainly weren't boring. I think that today I present a simple explanation of what then appeared as a not so simple concept. And I think that I can relate it to CSP.

1 Introduction

1.1 Physics and Societal Infrastructures

Physicists study that of nature which can be measured within us, around us and between 'within' and 'around'! To make mathematical models of physics phenomena, physics has helped develop and uses mathematics, notably calculus and statistics.

Domain engineers primarily studies societal infrastructure components which can be reasoned about, built and manipulated by humans. To make domain models of infrastructure components, domain engineering makes use of formal specification languages, their reasoning systems: formal testing, model checking and verification, and their tools.

Physicists turns to algebra in order to handle structures in nature. Algebra appears to be useful in a number of applications, to wit: the abstract modelling of chemical compounds. But there seems to be many structures in nature that cannot be captured in a satisfactory way by mathematics, including algebra and when captured in discrete mathematical disciplines such as sets, graph theory and combinatorics the "integration" of these mathematically represented — structures with calculus (etc.) — becomes awkward; it seems so much so that I know of no successful attempts.

Domain engineers turns to discrete mathematics — as embodied in formal specification languages and as "implementable" in programming languages — in order to handle structures in societal infrastructure components. These languages allow (a) the expression of arbitrarily complicated structures, (b) the evaluation of properties over such structures, (c) the "building & demolition" of such structures, and (d) the reasoning over such structures. They also allow the expression of dynamically varying structures —

something mathematics is "not so good at" ! But the specification languages have two problems: (i) they do not easily, if at all, handle continuity, that is, they do not embody calculus, or, for example, statistical concepts, etc., and (ii) they handle actual structures of societal infrastructure components and attributes of atomic and composite entities of these – usually by identical techniques thereby blurring what we think is an important distinction.

1.2 From Simple Entities to Processes

We shall first consider the structural components of societal infrastructures as **simple entities**, without considering any operations on these entities. In fact, in this paper we shall not consider operations on entities at all. This is possible, we claim, and in a sense in clear defiance of algebraic approaches — say as embodied in OO-methodologies — since, as we are claiming, that "world" of societal infrastructure components can be understood to quite some depth without considering their operations.

We shall then "map" parts and wholes into **processes** ! By an "ontological trick" we re-interpret simple entities as processes and their connections, i.e., how they are put together, as channels between processes.

It is all very simple, or, at least, we need to first make it simple before we complicate things. In this paper we will only present the easy picture.

1.3 Structure of This Paper

The rest of the paper is organised as follows. First, in Sect. 2, we give a first main, a meta-example, of syntactic aspects of a class of mereologies. It narrates and formalises an abstraction of what is here called 'parts': 'assemblies' and 'units'. That is, structures of units with connectors that may be used to provide connections between parts. So an assembly has a mereology represented by units and sub-assemblies and their actual connections.

In Sect. 3 we informally show that the assembly/unit structures of Sect. 2 indeed model structures of a variety of infrastructure components.

Then, in Sect. 4, we discuss concepts of atomic and composite simple entities. With atomic simple entities we associate attributes, and these may exhibit conceptual structures, and with composite simple entities we associate attributes, any number of simple sub-entities and their mereology. We discuss notational and semantic means of expressing attributes and their possible structures, and sub-entities, and their mereologies. And we relate our presentation to the wider concept of mereology.

Section 5 "performs" the ontological trick of mapping the assembly and unit entities and their connections exemplified in Sect. 2 into CSP processes

and channels, respectively — the second and last main — meta-example and now of semantic aspects of a class of mereologies.

The paper does not discuss relations between what is presented here and other approaches. As such we have renounced on the paper being a proper attempt at a proper scientific paper. We apologise.

2 A Syntactic Model of a Class of Mereologies

2.1 Systems, Assemblies, Units

We speak of systems as assemblies. From an assembly we can immediately observe a set of parts. Parts are either assemblies or units. We do not further define what assemblies and units are.

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

Parts observed from an assembly are said to be immediately embedded in, that is, within, that assembly. Two or more different parts of an assembly are said to be immediately adjacent to one another.

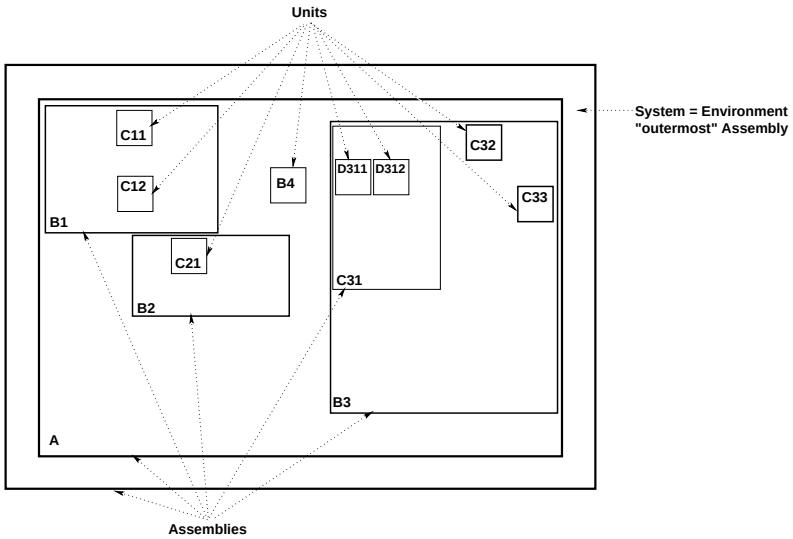

Fig. 1 Assemblies and Units "embedded" in an Environment

A system includes its environment. And we do not worry, so far, about the semiotics of all this !

Embeddedness and adjacency generalise to transitive relations.

Given obs_Ps we can define a function, xtr_Ps, which applies to an assembly a and which extracts all parts embedded in a and including a. The functions obs_Ps and xtr_Ps define the meaning of embeddedness.

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

union is the distributed union operator. Parts have unique identifiers. All parts observable from a system are distinct.

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

2.2 'Adjacency' and 'Within' Relations

Two parts, p,p′, are said to be *immediately next to*, i.e., i_next_to(p,p′)(a), one another in an assembly a if there exists an assembly, a' equal to or embedded in a such that p and p′ are observable in that assembly a′.

```
value
  i_next_to: P × P → A ~→ Bool, pre i_next_to(p,p′)(a): p≠p′
  i_next_to(p,p′)(a) ≡ ∃ a′:A • a′=a ∨ a′ ∈ xtr_Ps(a) • {p,p′}⊆obs_Ps(a′)
```

One part, p, is said to be *immediately within* another part, p′in an assembly a if there exists an assembly, a′ equal to or embedded in a such that p is observable in a′.

```
value
  i_within: P × P → A ~→ Bool
  i_within(p,p′)(a) ≡
    ∃ a′:A •  (a=a′ ∨ a′ ∈ xtr_Ps(a)) • p′=a′ ∧ p ∈ obs_Ps(a′)
```

We can generalise the immediate 'within' property. A part, p, is (transitively) within a part p′, within(p,p′)(a), of an assembly, a, either if p, is immediately within p′ of that assembly, a, or if there exists a (proper) part p″ of p′ such that within(p″,p)(a).

value
  within: P × P → A ~→ **Bool**
  within(p,p′)(a) ≡
    i_within(p,p′)(a) ∨ ∃ p″:P • p″ ∈ obs_Ps(p) ∧ within(p″,p′)(a)

The function within can be defined, alternatively, using xtr_Ps and i_within instead of obs_Ps and within :

value
  within′: P × P → A ~→ **Bool**
  within′(p,p′)(a) ≡
    i_within(p,p′)(a) ∨ ∃ p″:P • p″ ∈ xtr_Ps(p) ∧ i_within(p″,p′)(a)

lemma: within ≡ within′

We can generalise the immediate 'next to' property. Two parts, p, p′ of an assembly, a, are adjacent if they are either 'next to' one another or if there are two parts p_o, p'_o such that p, p′ are embedded in respectively p_o and p'_o and such that p_o, p'_o are immediately next to one another.

value
  adjacent: P × P → A ~→ **Bool**
  adjacent(p,p′)(a) ≡
    i_next_to(p,p′)(a) ∨
    ∃ p″,p‴:P • {p″,p‴}⊆xtr_Ps(a) ∧ i_next_to(p″,p‴)(a) ∧
      ((p=p″)∨within(p,p″)(a)) ∧ ((p′=p‴)∨within(p′,p‴)(a))

2.3 Mereology, Part I

So far we have built a *ground mereology* model, $\mathcal{M}_{\mathcal{G}_{\text{round}}}$. Let $\sqsubseteq$ denote *parthood, x is part of y*, $x \sqsubseteq y$.

$$\forall x(x \sqsubseteq x)^1 \tag{1}$$
$$\forall x, y(x \sqsubseteq y) \wedge (y \sqsubseteq x) \Rightarrow (x = y) \tag{2}$$
$$\forall x, y, z(x \sqsubseteq y) \wedge (y \sqsubseteq z) \Rightarrow (x \sqsubseteq z) \tag{3}$$

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

Let $\sqsubset$ denote *proper parthood*, *x is part of y*, $x \sqsubset y$. Formula 4 defines $x \sqsubset y$. Equivalence 5 can be proven to hold.

$$\forall x \sqsubset y =_{\text{def}} x(x \sqsubseteq y) \wedge \neg(x = y) \tag{4}$$

$$\forall\forall x, y(x \sqsubseteq y) \quad \Leftrightarrow \quad (x \sqsubset y) \vee (x = y) \tag{5}$$

The *proper part* $(x \sqsubset y)$ relation is a strict partial ordering:

$$\forall x \neg(x \sqsubset x) \tag{6}$$

$$\forall x, y(x \sqsubset y) \Rightarrow \neg(y \sqsubset x) \tag{7}$$

$$\forall x, y, z(x \sqsubset y) \wedge (y \sqsubset z) \Rightarrow (x \sqsubset z) \tag{8}$$

Overlap, $\bullet$, is also a relation of parts: Two individuals overlap if they have parts in common:

$$x \bullet y =_{\text{def}} \exists z(z \sqsubset x) \wedge (z \sqsubset y) \tag{9}$$

$$\forall x(x \bullet x) \tag{10}$$

$$\forall x, y(x \bullet y) \Rightarrow (y \bullet x) \tag{11}$$

Proper overlap, $\circ$, can be defined:

$$x \circ y =_{\text{def}} (x \bullet x) \wedge \neg(x \sqsubseteq y) \wedge \neg(y \sqsubseteq x) \tag{12}$$

Whereas Formulas (1-11) holds of the model of mereology we have shown so far, Formula (12) does not. In the next section we shall repair that situation.

The *proper part* relation, $\sqsubset$, reflects the *within* relation. The *disjoint* relation, $\oint$, reflects the *adjacency* relation.

$$x \oint y =_{\text{def}} \neg(x \bullet y) \tag{13}$$

Disjointness is symmetric:

$$\forall x, y(x \oint y) \Rightarrow (y \oint x) \tag{14}$$

The *weak supplementation* relation, Formula 15, expresses that if y is a proper part of x then there exists a part z such that z is a proper part of x and z and y are disjoint That is, whenever an individual has one proper part then it has more than one.

$$\forall x, y(y \sqsubset x) \ \Rightarrow \ \exists z(z \sqsubset x) \wedge (z \oint y) \tag{15}$$

Formulas 1–3 and 15 together determine the *minimal mereology*, $\mathcal{M}_{\mathcal{M}\text{inimal}}$. Formula 15 does not hold of the model of mereology we have shown so far. We shall comment on this in Sect. 4.2.

2.4 Connectors

So far we have only covered notions of parts being next to other parts or within one another. We shall now add to this a rather general notion of parts being otherwise related. That notion is one of connectors.

Connectors provide for connections between parts. A connector is an ability be be connected. A connection is the actual fulfillment of that ability. Connections are relations between pairs of parts. Connections "cut across" the "classical" *parts being part of the (or a) whole* and *parts being related by embeddedness or adjacency*.

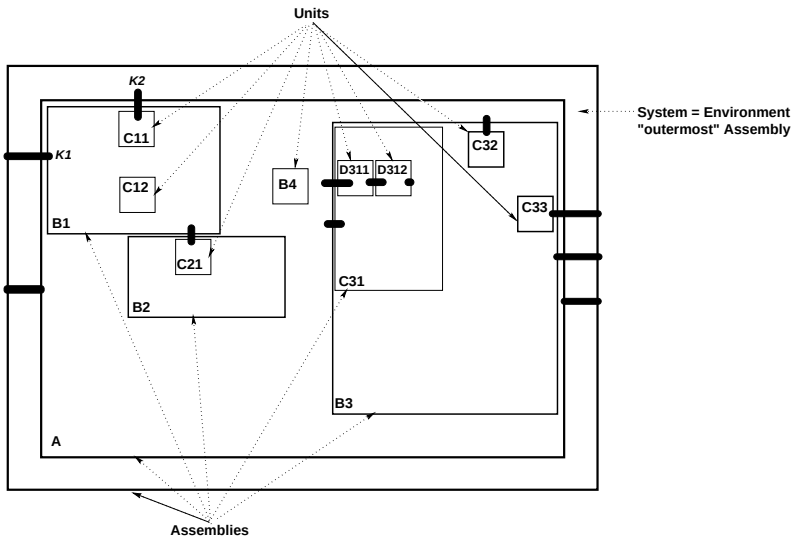

Fig. 2 Assembly and Unit Connectors: Internal and External

For now, we do not "ask" for the meaning of connectors !

Figure 2 "adds" connectors to Fig. 1 on page 4. The idea is that connectors allow an assembly to be connected to any embedded part, and allow two adjacent parts to be connected.

In Fig. 2 on the facing page the environment is connected, by *K2*, (without, as we shall later see, interfering with assemblies A and B1), to part C11; the "external world" is connected, by K1, to B1; etcetera. Later we shall discuss more general forms of connectors.

From a system we can observe all its connectors. From a connector we can observe its unique connector identifier and the set of part identifiers of the parts that the connector connects. All part identifiers of system connectors identify parts of the system. All observable connector identifiers of parts identify connectors of the system.

type
  K
value
  obs_Ks: S → K-**set**
  obs_KI: K → KI
  obs_Is: K → AUI-**set**
  obs_KIs: P → KI-**set**
axiom
  ∀ k:K • **card** obs_Is(k)=2,
  ∀ s:S,k:K • k ∈ obs_Ks(s) ⇒
    ∃ p:P • p ∈ xtr_Ps(s) ⇒ obs_AUI(p) ∈ obs_Is(k),
  ∀ s:S,p:P • ∀ ki:KI • ki ∈ obs_KIs(p) ⇒
    ∃! k:K • k ∈ obs_Ks(s) ∧ ki=obs_KI(k)

This model allows for a rather "free-wheeling" notion of connectors one that allows internal connectors to "cut across" embedded and adjacent parts; and one that allows external connectors to "penetrate" from an outside to any embedded part.

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

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

2.5 Mereology, Part II

We shall interpret connections as follows: A connection between parts p_i and p_j that enjoy a p_i adjacent to p_j relationship, means $p_i \circ p_j$, that is, although parts p_i and p_j are adjacent they do *share* "something", i.e., have something *in common*. What that "something" is we shall comment on in Sect. 5.4. A connection between parts p_i and p_j that enjoy a p_i within p_j relationship, does not add other meaning than commented upon in Sect. 5.4 on page 22.

With the above interpretation we may arrive at the following, perhaps somewhat "awkward-looking" case: a connection connects two adjacent parts p_i and p_j where part p_i is within part p_{i_o} and part p_j is within part p_{j_o} where parts p_{i_o} and p_{j_o} are adjacent but not otherwise connected. How are we to explain that ! Since we have not otherwise interpreted the meaning of parts, we can just postulate that "so it is" ! We shall, in Sect. 5.4 on page 22, give a more satisfactory explanation.

In Sect. 2.3 we introduced the following operators: $\sqsubseteq, \sqsubset, \bullet, \circ$, and $\oint$ In some of the mereology literature [13–15] these operators are symbolised with caligraphic letters: $\sqsubseteq$: $\mathcal{P}$: part, $\sqsubset$: $\mathcal{PP}$: proper part, $\bullet : \mathcal{O}$: overlap and $\oint : \mathcal{U}$: underlap.

2.6 Discussion

2.6.1 Summary

This ends our first model of a concept of mereology. The parts are those of assemblies and units. The relations between parts and the whole are, on one hand, those of embeddedness i.e. within, and adjacency, i.e., adjacent, and on the other hand, those expressed by connectors: relations between arbitrary parts and between arbitrary parts and the exterior.

2.6.2 Extensions

A number of extensions are possible: one can add "mobile" parts and "free" connectors, and one can further add operations that allow such mobile parts to move from one assembly to another along routes of connectors. Free connectors and mobility assumes static versus dynamic parts and connectors: a free connector is one which allows a mobile part to be connected to another part, fixed or mobile; and the potentiality of a move of a mobile part introduces a further dimension of dynamics of a mereology.

2.6.3 Comments

We shall leave the modelling of free connectors and mobile parts to another time. Suffice it now to indicate that the mereology model given so far is relevant: that it applies to a somewhat wide range of application domain structures, and that it thus affords a uniform treatment of proper formal models of these application domain structures.

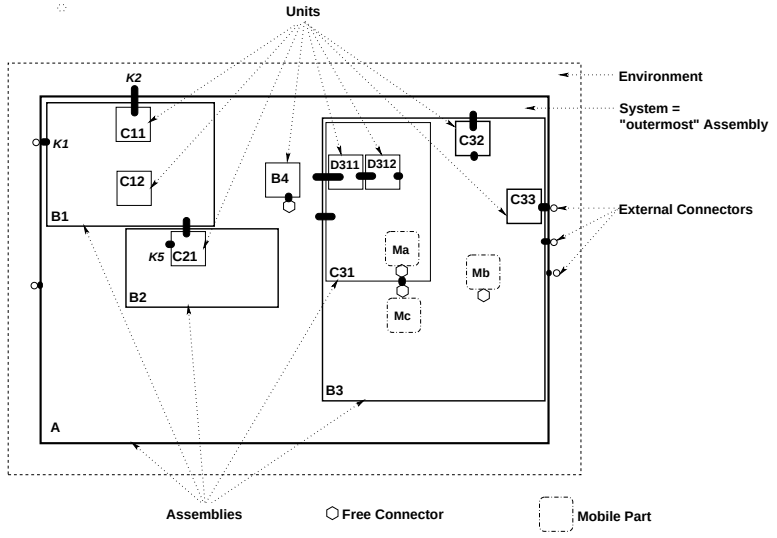

Fig. 3 Mobile Parts and Free Connectors

3 Discussion & Interpretation

Before a semantic treatment of the concept of mereology let us review what we have done and let us interpret our abstraction (i.e., relate it to actual societal infrastructure components).

3.1 What We have Done So Far ?

We have presented a model that is claimed to abstract essential mereological properties of machine assemblies, railway nets, the oil industry, oil pipelines, buildings and their installations, hospitals, etcetera.

3.2 Six Interpretations

Let us substantiate the claims made in the previous paragraph. We will do so, albeit informally, in the next many paragraphs. Our substantiation is a form of diagrammatic reasoning. Subsets of diagrams will be claimed to represent parts, while Other subsets will be claimed to represent connectors. The reasoning is incomplete.

3.2.1 Air Traffic

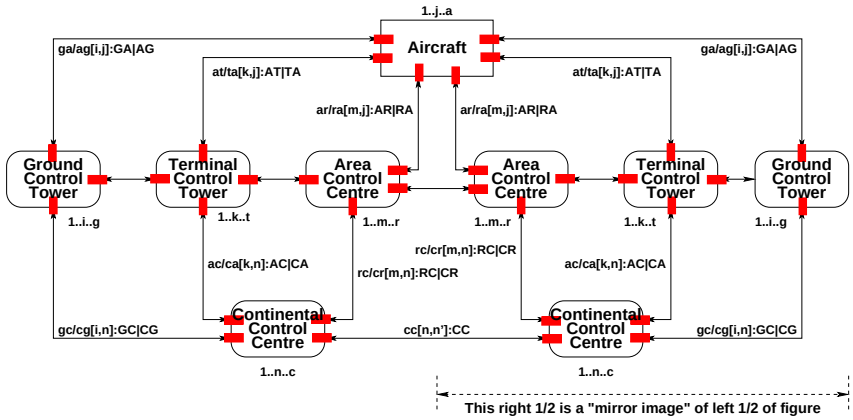

Fig. 4 An air traffic system. Black boxes and lines are units; red boxes are connections

Figure 4 shows nine (9) boxes and eighteen (18) lines. Together they form an assembly. Individually boxes and lines represent units. The rounded corner boxes denote buildings. The sharp corner box denote an aircraft. Lines denote radio telecommunication. Only where lines touch boxes do we have connections. These are shown as red horisontal or vertical boxes at both ends of the double-headed arrows, overlapping both the 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) unit shown. Notice that the 'box' units 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 line for not overlapping. And if they were overlapping, then that must have been a decision of the air traffic system.

3.2.2 Buildings

Figure 5 on the facing page shows a building plan — as an assembly of two neighbouring, common wall-sharing buildings, A and H, probably built at different times; with room sections B, C, D and E contained within A, and room sections I, J and K within H; with room sections L and M within K, and F and G within C. Connector γ provides means of a connection between A and B. Connection κ provides "access" between B and F. Connectors ι and ω enable input, respectively output adaptors (receptor, resp. outlet) for

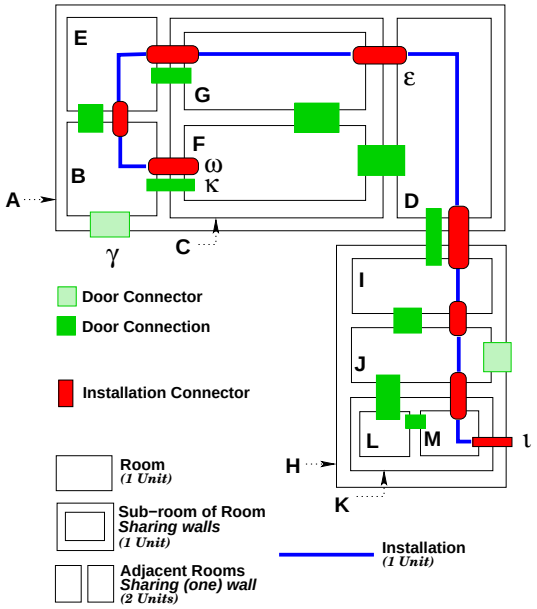

Fig. 5 A building plan with installation

electricity (or water, or oil), connection ϵ allow electricity (or water, or oil) to be conducted through a wall. Etcetera.

3.2.3 Financial Service Industry

Figure 6 on the next page shows seven (7) larger boxes [6 of which are shown by dashed lines] and twelve (12) double-arrowed lines. Where double-arrowed lines touch upon (dashed) boxes we have connections (also to inner boxes). Six (6) of the boxes, the dashed line boxes, are assemblies, five (5) of them consisting of a variable number of units; five (5) are here shown as having three units each with bullets "between" them to designate "variability". People, not shown, access the outermost (and hence the "innermost" boxes, but the latter is not shown) through connectors, shown by bullets, •.

3.2.4 Machine Assemblies

Figure 7 on the following page shows a machine assembly. Square boxes show assemblies or units. Bullets, •, show connectors. Strands of two or three bullets on a thin line, encircled by a rounded box, show connections. The full, i.e., the level 0, assembly consists of four parts and three internal and three

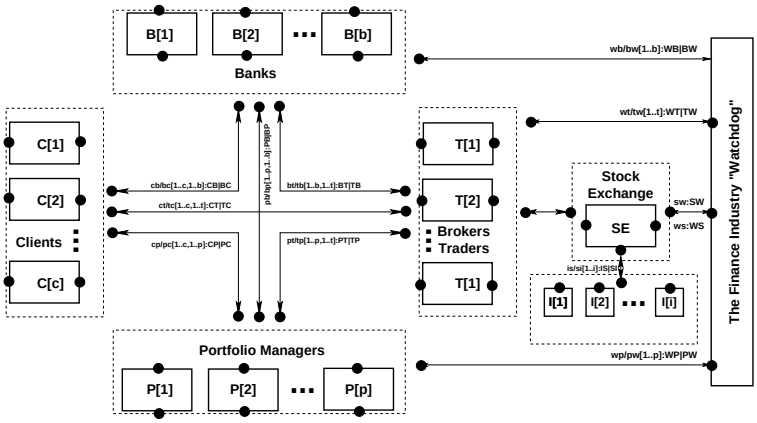

Fig. 6 A financial service industry

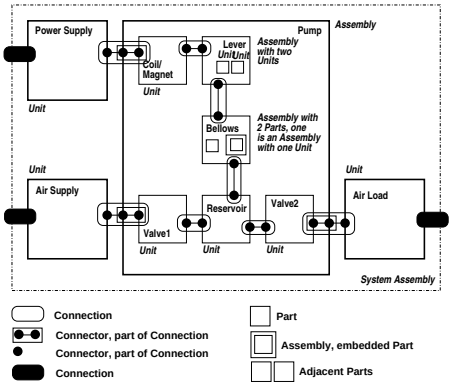

Fig. 7 An air pump, i.e., a physical mechanical system

external connections. The Pump unit is an assembly of six (6) parts, five (5) internal connections and three (3) external connectors. Etcetera. One connector and some connections afford "transmission" of electrical power. Other connections convey torque. Two connectors convey input air, respectively output air.

3.2.5 Oil Industry

"The" Overall Assembly

Figure 8 on the next page shows an assembly consisting of fourteen (14) assemblies, left-to-right: one oil field, a crude oil pipeline system, two refineries

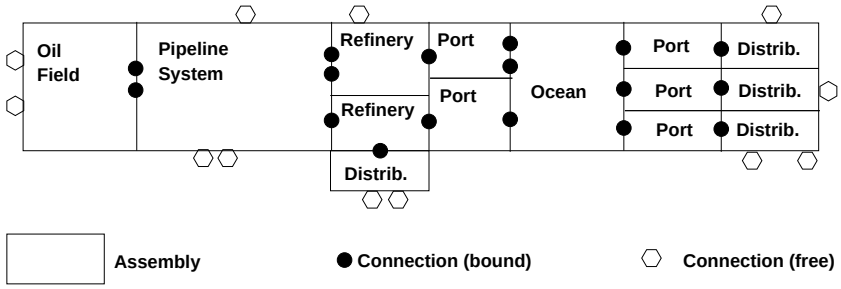

Fig. 8 A Schematic of an Oil Industry

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 assembly units there are connections, and from some of the assembly units there are connectors (to an external environment). The crude oil pipeline system assembly unit will be concretised next.

A Concretised Assembly Unit

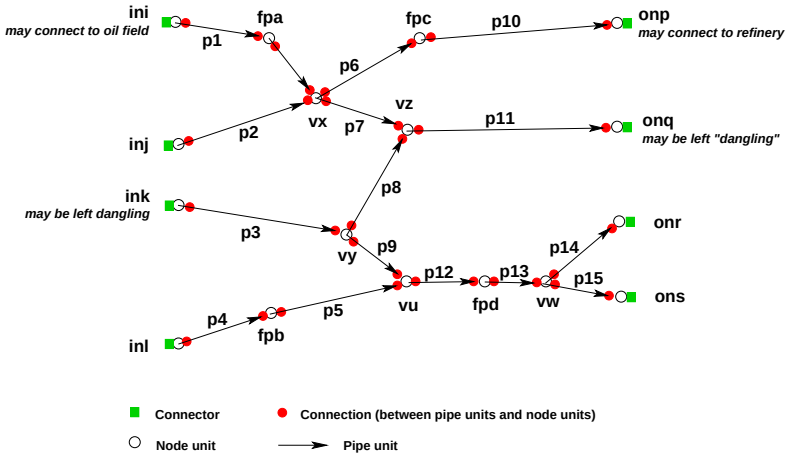

Fig. 9 A Pipeline System

Figure 9 shows a pipeline system. It consists of 32 units: 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) 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), 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 red[2] disc connections. Input and output nodes have input, respectively output connectors, one each, and shown with green[3]

3.2.6 Railway Nets

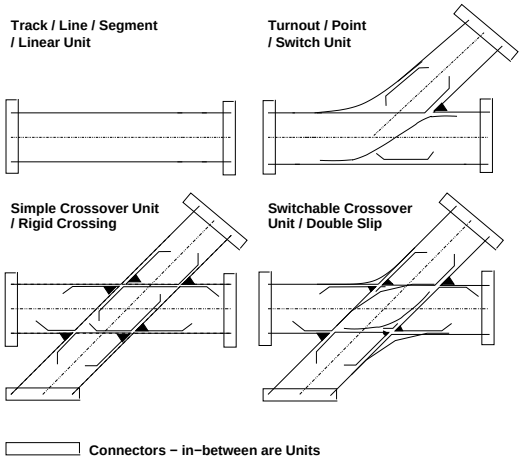

Fig. 10 Four example rail units

Figure 10 diagrams four rail units, each with their two, three or four connectors. Multiple instances of these rail units can be assembled as shown on Fig. 11 on the facing page into proper rail nets.

Figure 11 on the next page 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 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.

[2] This paper is most likely not published with colours, so red will be shown as darker colour.

[3] Shown as lighter coloured connections.

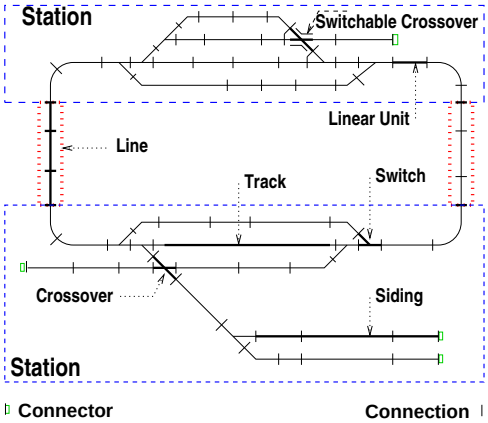

Fig. 11 A "model" railway net. An Assembly of four Assemblies: Two stations and two lines; Lines here consist of linear rail units; stations of all the kinds of units shown in Fig. 10 on the facing page. There are 66 connections and four "dangling" connectors

3.3 Discussion

It requires a somewhat more laborious effort, than just "flashing" and commenting on these diagrams, to show that the modelling of essential aspects of their structures can indeed be done by simple instantiation of the model given in the previous section. We can refer to a number of documents which give rather detailed domain models of air traffic [1], container line industry [9][4], financial service industry (banks, credit card companies, brokers, traders and securities and commodities exchanges, insurance companies, etc.)[5], healthcare [18, Sects. 10.2.2 + 10.4.2], IT security [19], "the market" (consumers, retailers, wholesalers, producers and distribution chains) [2], "the" oil industry[6], transportation nets[7], railways [3, 4, 38, 39, 46] and [18, Sect. 10.6][8], etcetera, etcetera. Seen in the perspective of the present paper we claim that much of the modelling work done in those references can now be considerably shortened and trust in these models correspondingly increased.

[4] http://www2.imm.dtu.dk/~db/container-paper.pdf

[5] http://www2.imm.dtu.dk/~db/fsi.pdf

[6] http://www2.imm.dtu.dk/~db/pipeline.pdf

[7] http://www2.imm.dtu.dk/~db/transport.pdf

[8] http://www.railwaydomain.org/

4 Simple Entities

The reason for our interest in 'simple entities' is that assemblies and units of systems possess static and dynamic properties which become contexts and states of the processes into which we shall "transform" simple entities.

4.1 Observable Phenomena

We shall just consider 'simple entities'.[9] By a simple entity we shall here understand a phenomenon that we can designate, viz. see, touch, hear, smell or taste, or measure by some instrument (of physics, incl. chemistry). A simple entity thus has properties. A simple entity is either continuous or is discrete, and then it is either atomic or composite.

4.1.1 Attributes: Types and Values

By an attribute we mean a simple property of an entity. *A simple entity has properties* $p_i, p_j, \ldots, p_k$. Typically we express attributes by a pair of a type designator: *the attribute is of type* V, and a value: *the attribute has value* v (of type V, i.e., $v : V$). A simple entity may have many simple properties. A continuous entity, like 'oil', may have the following attributes: type: *petroleum*, kind: *Brent-crude*, amount: *6 barrels*, price: *45 US $/barrel*. An *atomic* entity, like a 'person', may have the following attributes: gender: *male*, name: *Dines Bjørner*, birth date: *4. Oct. 1937*, marital status: *married*. A *composite* entity, like a railway system, may have the following attributes: country: *Denmark*, name: *DSB*, electrified: *partly*, owner: *independent public enterprise owned by Danish Ministry of Transport.*

4.1.2 Continuous Simple Entities

A simple entity is said to be continuous if, within limits, reasonably sizable amounts of the simple entity, can be arbitrarily decomposed into smaller parts each of which still remain simple continuous entities of the same simple entity kind. Examples of continuous entities are: oil, i.e., any fluid, air, i.e., any gas, time period and a measure of fabric.

[9] We use use the name 'simple entities' in contrast to 'entities' which we see as comprising all of simple entities, functions, events and behaviours. "Interesting" functions and normal events involve all forms of entities.

4.1.3 Discrete Simple Entities

A simple entity is said to be discrete if its immediate structure is not continuous. A simple discrete entity may, however, contain continuous sub-entities. Examples of discrete entities are: persons, rail units, oil pipes, a group of persons, a railway line and an oil pipeline.

Atomic Simple Entities

A simple entity is said to be atomic if it cannot be meaningfully decomposed into parts where these parts has a useful "value" in the context in which the simple entity is viewed and while still remaining an instantiation of that entity. Thus a 'physically able person', which we consider atomic, can, from the point of physical ability, not be decomposed into meaningful parts: a leg, an arm, a head, etc. Other atomic entities could be a rail unit, an oil pipe, or a hospital bed. The only thing characterising an atomic entity are its attributes.

Composite Simple Entities

A simple entity, c, is said to be composite if it can be meaningfully decomposed into sub-entities that have separate meaning in the context in which c is viewed. We exemplify some composite entities. (1) A *railway net* can be decomposed into a set of one or more *train lines* and a set of two or more *train stations*. Lines and stations are themselves composite entities. (2) An *Oil industry* whose decomposition include: one or more *oil fields*, one or more *pipeline systems*, one or more *oil refineries* and one or more *one or more oil product distribution systems*. Each of these sub-entities are also composite. Composite simple entities are thus characterisable by their attributes, their sub-entities, and the mereology of how these sub-entities are put together.

4.2 Mereology, Part III

Formula 15 on page 8 expresses that whenever an individual has one proper part then it has more than one. We mentioned there, Page 8, that we would comment on the fact that our model appears to allow that assemblies may have just one proper part. We now do so. We shall still allow assemblies to have just one proper part — in the sense of a sub-assembly or a unit — but we shall interpret the fact that an assembly always have at least one attribute. Therefore we shall "generously" interpret the set of attributes of an assembly to constitute a part. In Sect. 5 we shall see how attributes of both units and

assemblies of the interpreted mereology contribute to the state components of the unit and assembly processes.

4.3 Discussion

In Sect. 3.2 we interpreted the model of mereology in six examples. The units of Sect. 2 which in that section were left uninterpreted now got individuality — in the form of aircraft, building rooms, rail units and oil pipes. Similarly for the assemblies of Sect. 2. They became pipeline systems, oil refineries, train stations, banks, etc.In conventional modelling the mereology of an infrastructure component, of the kinds exemplified in Sect. 3.2, was modelled by modelling that infrastructure component's special mereology together, "in line", with the modelling of unit and assembly attributes. With the model of Sect. 2 now available we do not have to model the mereological aspects, but can, instead, instantiate the model of Sect. 2 appropriately. We leave that to be reported upon elsewhere. In many conventional infrastructure component models it was often difficult to separate what was mereology from what were attributes.

5 A Semantic Model of a Class of Mereologies

5.1 The Mereology Entities ≡ Processes

The model of mereology presented in Sect. 2 (Pages 4–10) focused on the following simple entities (i) the assemblies, (ii) the units and (iii) the connectors. To assemblies and units we associate `CSP` processes, and to connectors we associate a `CSP` channels, one-by-one [32, 33, 41, 43]. The connectors form the mereological attributes of the model.

5.2 Channels

The `CSP` channels, are each "anchored" in two parts: if a part is a unit then in "its corresponding" unit process, and if a part is an assembly then in "its corresponding" assembly process. From a system assembly we can extract all connector identifiers. They become indexes into an array of channels. Each of the connector channel identifiers is mentioned in exactly two unit or assembly processes.

value

```
  s:S
  kis:KI-set = xtr∀KIs(s)
type
  ChMap = AUI ~m~> KI-set
value
  cm:ChMap = [ obs_AUI(p)↦obs_KIs(p)|p:P•p ∈ xtr_Ps(s) ]
channel
  ch[ i|i:KI•i ∈ kis ] MSG
```

5.3 Process Definitions

```
value
  system: S → Process
  system(s) ≡ assembly(s)

  assembly: a:A→in,out {ch[ cm(i) ]|i:KI•i ∈ cm(obs_AUI(a))} process
  assembly(a) ≡
    Mℳ(a)(obs_Aℰ(a)) ‖
    ‖ {assembly(a′)|a′:A•a′ ∈ obs_Ps(a)} ‖
    ‖ {unit(u)|u:U•u ∈ obs_Ps(a)}
  obs_Aℰ: A → Aℰ

  Mℳ: a:A→Aℰ→in,out {ch[ cm(i) ]|i:KI•i ∈ cm(obs_AUI(a))} process
  Mℳ(a)(aσ) ≡ Mℳ(a)(Aℱ(a)(aσ))

  Aℱ: a:A → Aℰ → in,out {ch[ em(i) ]|i:KI•i ∈
  cm(obs_AUI(a))}×Aℰ

  unit: u:U → in,out {ch[ cm(i) ]|i:KI•i ∈ cm(obs_UI(u))} process
  unit(u) ≡ Mℳ(u)(obs_Uℰ(u))
  obs_Uℰ: U → Uℰ

  Mℳ: u:U → Uℰ → in,out {ch[ cm(i) ]|i:KI•i ∈ cm(obs_UI(u))} process
  Mℳ(u)(uσ) ≡ Mℳ(u)(Uℱ(u)(uσ))

  Uℱ: U → Uℰ → in,out {ch[ em(i) ]|i:KI • i ∈ cm(obs_AUI(u))}  Uℰ
```

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

5.4 Mereology, Part III

A little more meaning has been added to the notions of parts and connections. The within and adjacent to relations between parts (assemblies and units) reflect a phenomenological world of geometry, and the connected relation between parts (assemblies and units) 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".

5.5 Discussion

5.5.1 Partial Evaluation

The assembly function "first" "functions" as a compiler. The 'compiler' translates an assembly structure into three process expressions: the $\mathcal{M}_{\mathcal{A}}(a)(a\sigma)$ invocation, the parallel composition of assembly processes, a', one for each sub-assembly of a, and the parallel composition of unit processes, one for each unit of assembly a — with these three process expressions "being put in parallel". The recursion in assembly ends when a sub-...-assembly consists of no sub-sub-...-assemblies. Then the compiling task ends and the many generated $\mathcal{M}_{\mathcal{A}}(a)(a\sigma)$ and $\mathcal{M}_{\mathcal{U}}(u)(u\sigma)$ process expressions are invoked.

5.5.2 Generalised Channel Processes

We can refine the meaning of connectors. Each connector, so far, was modelled by a CSP channel. CSP channels serve both as a synchronisation and as a communication medium. We now suggest to model it by a process. A channel process can be thought of as having four channels and a buffering process. Connector, κ:K, may connect parts π_i, π_j. The four channels could be thought of as indexed by $(\kappa, \pi_i), (\pi_i, \kappa), (\kappa, \pi_j)$ and (π_j, κ). The process buffer could, depending on parts p_i, p_j, be either queues, sets, bags, stacks, or other.

6 Conclusion

6.1 Summary

We have proposed a simple model which we claim captures a large variety of structures of societal infrastructure components (Sect. 2). The model focused on parts, their within and next to one another relations as well as connections between parts. We have, rather briefly, held that model up against a variety of diagrammatic renditions of specific societal infrastructure components (Sect. 3) and claimed that the model is relevant for their formalisation. We have then reviewed the concepts of continuous (fluid, gaseous) and discrete (fixed, solid) simple entities and especially discussed the discrete atomic and composite simple entities (Sect. 4) and their attributes and sub-entities. We have done so in order first to [again] single out the topic of the mereology of composite (discrete) entities, and then to prepare for the next section's process states (and environments) – modelled from simple entity attributes. We have finally shown how one can relate simple entities to CSP processes and connectors to CSP channels (Sect. 5).

6.2 What Have We Achieved ?

There is, as we indicated, in Sect. 3, a bewildering variety of from societal infrastructure component to "gadget" structures – and these structures must be modelled. We claim that the mereology model (of Sect. 2) provides a common denominator for all of these: that the model is generic and can be simply instantiated for each of the shown, and, we again claim, for many other domain examples. We claim that the model (of Sect. 2) can serve as a basis for investigating the axiom systems proposed for mereology [13, Casati & Varzi] and a calculus of individuals [14, Bowman L. Clarke]. We thus claim to have a simple model for the kind of mereologies presented in the literature.

6.3 Open Points

We have yet to carefully demonstrate two classes of things: (i) to properly refine our mereology model into models for the sub-entity structures of specific societal infrastructure components etc.; and (ii) to identify the exact relations between our model of mereology and the axiom systems presented in the literature [13, 14].

6.4 The Memorial and The Laudatio

On Douglas Taylor Ross:

> *It is possible his work in that direction became too pioneering or too advanced for his colleagues, including us. Who knows, the future may prove him right. At any rate, his reflections regularly made me think.*
>
> Michel Sintzoff, 2007

Fig. 12 Doug Taylor Ross and Sir Charles Anthony Richard Hoare

6.5 Acknowledgements

I thank University of Saarland for hosting me during some of the time when I wrote this paper.

7 Bibliographical Notes

The present paper uses the RAISE Specification Language [5–7, 27, 28, 30]. The concept of mereology appears to have been first studied by Stanisław Leśniewski [35,45]. Seminal mereology papers appears to be [13,14,34]. Since the present paper was first written and presented, April 16, 2009, and its revision for publication,I have thought more about the mereological issues and, at the instigation of Tony Hoare, combined these with a study of Bertrand Russell's *Philosophy of Logical Atomism* [42] and [44, Vol. 8, Part III, Chap. 17, pp 157–244]. The outcome became [8].

References

1. Dines Bjørner. Software Systems Engineering — From Domain Analysis to Requirements Capture: An Air Traffic Control Example. In *2nd Asia-Pacific Software Engineering Conference (APSEC '95)*. IEEE Computer Society, 6–9 December 1995. Brisbane, Queensland, Australia.
2. Dines Bjørner. Domain Models of "The Market" — in Preparation for E–Transaction Systems. In *Practical Foundations of Business and System Specifications (Eds.: Haim Kilov and Ken Baclawski)*, The Netherlands, December 2002. Kluwer Academic Press.
3. Dines Bjørner. Dynamics of Railway Nets: On an Interface between Automatic Control and Software Engineering. In *CTS2003: 10th IFAC Symposium on Control in Transportation Systems*, Oxford, UK, August 4-6 2003. Elsevier Science Ltd. Symposium held at Tokyo, Japan. Editors: S. Tsugawa and M. Aoki.
4. Dines Bjørner. New Results and Trends in Formal Techniques for the Development of Software for Transportation Systems. In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*. Institut für Verkehrssicherheit und Automatisierungstechnik, Techn.Univ. of Braunschweig, Germany, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.
5. Dines Bjørner. *Software Engineering, Vol. 1: Abstraction and Modelling*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
6. Dines Bjørner. *Software Engineering, Vol. 2: Specification of Systems and Languages*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006. Chapters 12–14 are primarily authored by Christian Krog Madsen.
7. Dines Bjørner. *Software Engineering, Vol. 3: Domains, Requirements and Software Design*. Texts in Theoretical Computer Science, the EATCS Series. Springer, 2006.
8. Dines Bjørner. An Emerging Domain Science – A Rôle for Stanisław Leśniewski's Mereology and Bertrand Russell's Philosophy of Logical Atomism. *Higher-order and Symbolic Computation*, 2009.
9. Dines Bjørner. Domain Engineering. In *The 2007 Lipari PhD Summer School*, Lecture Notes in Computer Science (eds. E. Börger and A. Ferro), pages 1–102, Heidelberg, Germany, 2009. Springer. To appear. Meanwhile check with http://www2.imm.dtu.dk/~db/container-paper.pdf.
10. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*. JAIST Press, March 2009. The monograph contains the following chapters: [17–26].
11. Dines Bjørner and Martin C. Henson, editors. *Logics of Specification Languages*. EATCS Series, Monograph in Theoretical Computer Science. Springer, Heidelberg, Germany, 2008.
12. Dominique Cansell and Dominique Méry. Logical Foundations of the B Method. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [16,29,31,36,37,40] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
13. R. Casati and A. Varzi. *Parts and Places: the structures of spatial representation*. MIT Press, 1999.
14. Bowman L. Clarke. A Calculus of Individuals Based on 'Connection'. *Notre Dame J. Formal Logic*, 22(3):204–218, 1981.
15. Bowman L. Clarke. Individuals and Points. *Notre Dame J. Formal Logic*, 26(1):61–75, 1985.
16. Ražvan Diaconescu, Kokichi Futatsugi, and Kazuhiro Ogata. CafeOBJ: Logical Foundations and Methodology. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [12, 29, 31, 36, 37, 40] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.

17. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering*, chapter 5: The Triptych Process Model – Process Assessment and Improvement, pages 107–138. JAIST Press, March 2009.
18. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [10]*, chapter 10: Towards a Family of Script Languages – – Licenses and Contracts – Incomplete Sketch, pages 283–328. JAIST Press, March 2009.
19. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [10]*, chapter 9: Towards a Model of IT Security — – The ISO Information Security Code of Practice – An Incomplete Rough Sketch Analysis, pages 223–282. JAIST Press, March 2009.
20. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [10]*, chapter 1: On Domains and On Domain Engineering – Prerequisites for Trustworthy Software – A Necessity for Believable Management, pages 3–38. JAIST Press, March 2009.
21. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [10]*, chapter 2: Possible Collaborative Domain Projects – A Management Brief, pages 39–56. JAIST Press, March 2009.
22. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [10]*, chapter 3: The Rôle of Domain Engineering in Software Development, pages 57–72. JAIST Press, March 2009.
23. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [10]*, chapter 4: Verified Software for Ubiquitous Computing – A VSTTE Ubiquitous Computing Project Proposal, pages 73–106. JAIST Press, March 2009.
24. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [10]*, chapter 6: Domains and Problem Frames – The Triptych Dogma and M.A.Jackson's PF Paradigm, pages 139–175. JAIST Press, March 2009.
25. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [10]*, chapter 7: Documents – A Rough Sketch Domain Analysis, pages 179–200. JAIST Press, March 2009.
26. Dines Bjørner. *Domain Engineering: Technology Management, Research and Engineering [10]*, chapter 8: Public Government – A Rough Sketch Domain Analysis, pages 201–222. JAIST Press, March 2009.
27. Chris George and Anne E. Haxthausen. *Logics of Specification Languages*, chapter The Logic of the RAISE Specification Language, pages 349–399 in [11]. Springer, 2008.
28. Chris W. George, Peter Haff, Klaus Havelund, Anne Elisabeth Haxthausen, Robert Milne, Claus Bendix Nielsen, Søren Prehn, and Kim Ritter Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.
29. Chris W. George and Anne E. Haxthausen. The Logic of the RAISE Specification Language. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [12, 16,31,36,37,40] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
30. Chris W. George, Anne Elisabeth Haxthausen, Steven Hughes, Robert Milne, Søren Prehn, and Jan Storbank Pedersen. *The RAISE Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.
31. Martin C. Henson, Steve Reeves, and Jonathan P. Bowen. Z Logic and its Consequences. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [12, 16, 29, 36, 37, 40] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
32. Tony Hoare. *Communicating Sequential Processes*. C.A.R. Hoare Series in Computer Science. Prentice-Hall International, 1985.
33. Tony Hoare. Communicating Sequential Processes. Published electronically: `http://www.usingcsp.com/cspbook.pdf`, 2004. Second edition of [32]. See also `http://www.usingcsp.com/`.

34. Henry S. Leonard and Nelson Goodman. The Calculus of Individuals and its Uses. *Journal of Symbolic Logic*, 5:45–44, 1940.
35. E.C. Luschei. *The Logical Systems of Leśniewksi*. North Holland, Amsterdam, The Netherlands, 1962.
36. Stephan Merz. On the Logic of TLA+. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [12, 16, 29, 31, 37, 40] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
37. Till Mossakowski, Anne E. Haxthausen, Don Sanella, and Andzrej Tarlecki. CASL — The Common Algebraic Specification Language: Semantics and Proof Theory. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [12,16,29,31,36,40] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
38. Martin Pěnička and Dines Bjørner. From Railway Resource Planning to Train Operation — a Brief Survey of Complementary Formalisations. In *Building the Information Society, IFIP 18th World Computer Congress, Topical Sessions, 22–27 August, 2004, Toulouse, France — Ed. Renéne Jacquart*, pages 629–636. Kluwer Academic Publishers, August 2004.
39. Martin Pěnička, Albena Kirilova Strupchanska, and Dines Bjørner. Train Maintenance Routing. In *FORMS'2003: Symposium on Formal Methods for Railway Operation and Control Systems*. L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.
40. Wolfgang Reisig. The Expressive Power of Abstract State Machines. *Computing and Informatics*, 22(1–2), 2003. This paper is one of a series: [12,16,29,31,36,37] appearing in a double issue of the same journal: *Logics of Specification Languages* — edited by Dines Bjørner.
41. A. W. Roscoe. *Theory and Practice of Concurrency*. C.A.R. Hoare Series in Computer Science. Prentice-Hall, 1997. Now available on the net: http://www.comlab.ox.ac.uk/-people/bill.roscoe/publications/68b.pdf.
42. Bertrand Russell. The Philosophy of Logical Atomism. *The Monist: An International Quarterly Journal of General Philosophical Inquiry,*, xxxviii–xxix:495–527, 32–63, 190–222, 345–380, 1918–1919.
43. Steve Schneider. *Concurrent and Real-time Systems — The CSP Approach*. Worldwide Series in Computer Science. John Wiley & Sons, Ltd., Baffins Lane, Chichester, West Sussex PO19 1UD, England, January 2000.
44. John G. Slater, editor. *The Collected Papers of Bertrand Russel*. Allen and Unwin, London, England, 1986.
45. J.T.J. Srzednicki and Z. Stachniak, editors. *Leśniewksi's Lecture Notes in Logic*. Dordrecht, 1988.
46. Albena Kirilova Strupchanska, Martin Pěnička, and Dines Bjørner. Railway Staff Rostering. In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems*. L'Harmattan Hongrie, 15–16 May 2003. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.