A Rôle for Mereology in Domain Science and Engineering

– to every mereology there corresponds a λ–expression

Dines Bjørner

Bergen, 9 May 2012

DTU Informatics, May 1, 2012: 19:06

0. Summary

- We give an abstract model of parts and part-hood relations
- of software application domains such as
  - the financial service industry,
  - railway systems,
  - road transport systems,
  - health care,
  - oil pipelines,
  - secure [IT] systems,

  etcetera.
- We relate this model
  - to axiom systems for mereology, showing satisfiability, and
  - show that for every mereology there corresponds a class of `Communicating Sequential Processes`,
- that is: a λ–expression.

1. Introduction

- The term ‘mereology’ is accredited to the Polish mathematician, philosopher and logician Stansław Leśniewski (1886–1939) who
  - “was a nominalist: he rejected axiomatic set theory
  - and devised three formal systems,
    - *Protothetic*,
    - *Ontology*, and
    - *Mereology*

  as a concrete alternative to set theory”.
- In this seminar I shall be concerned with only
  - certain aspects of mereology,
  - namely those that appears most immediately relevant to domain science
  - (a relatively new part of current computer science).

1.1. Computing Science Mereology

- "Mereology (from the Greek $\mu\epsilon\rho o\varsigma$ 'part') is the theory of parthood relations: of the relations of part to whole and the relations of part to part within a whole"[1].
- In this talk we restrict 'parts' to be those that,
  - firstly, are spatially distinguishable, then,
  - secondly, while "being based" on such spatially distinguishable parts, are conceptually related.
- The relation: "being based", shall be made clear in this talk.

[1] Achille Varzi: Mereology, http://plato.stanford.edu/entries/mereology/ 2009 and [CasatiVarzi1999]

- Accordingly two parts, p_x and p_y, (of a same "whole") are
  - are either "adjacent",
  - or are "embedded within" one another

  as loosely indicated in Fig. 1.

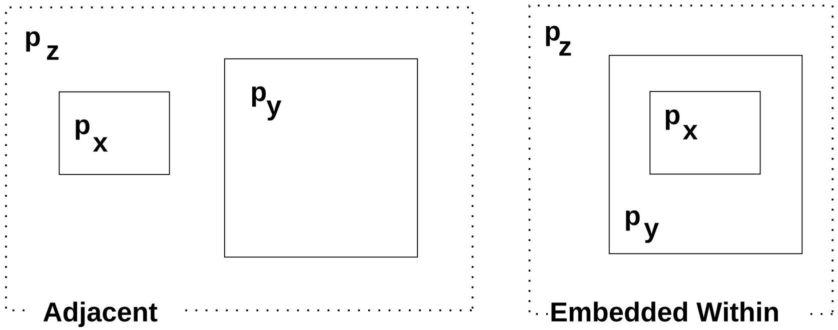

Figure 1: 'Adjacent' and "Embedded Within' parts

- ‘Adjacent’ parts
  - are direct parts of a same third part, p_z,
  - i.e., p_x and p_y are “embedded within” p_z;
  - or one (p_x) or the other (p_y) or both (p_x and p_y) are parts of a same third part, p'_z “embedded within” p_z;
  - etcetera;

  as loosely indicated in Fig. 2 on the next slide.
- or one is “embedded within” the other — etc.
  as loosely indicated in Fig. 2 on the facing slide.

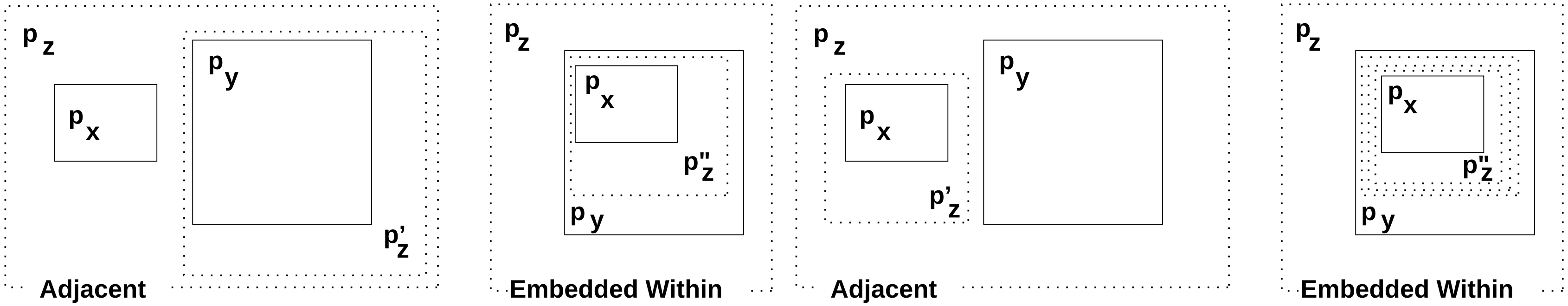

Figure 2: ‘Adjacent’ and “Embedded Within’ parts

- Parts, whether adjacent or embedded within one another, can share properties.
  - For adjacent parts this sharing seems, in the literature, to be diagrammatically expressed by letting the part rectangles “intersect”.
  - Usually properties are not spatial hence ‘intersection’ seems confusing.
  - We refer to Fig. 3 on the next slide.

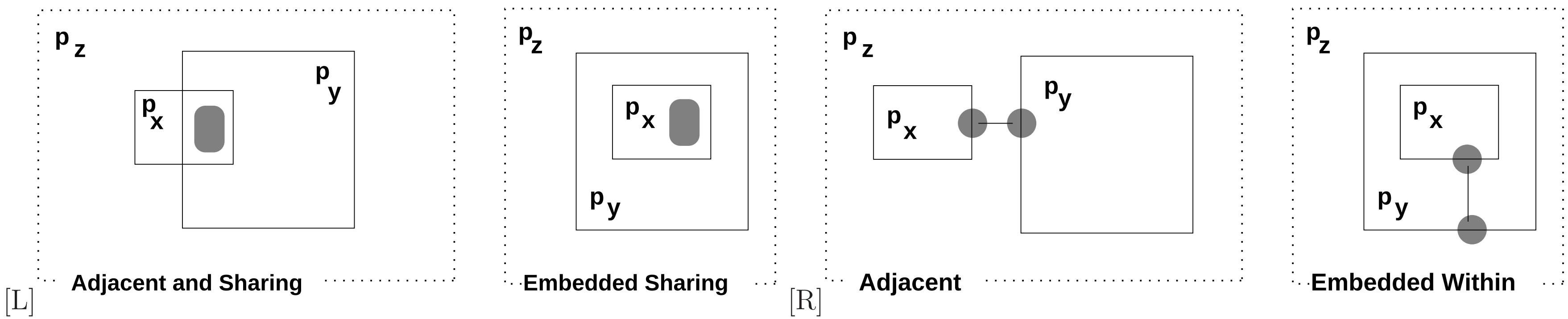

Figure 3: Two models, [L,R], of parts sharing properties

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

1.2. From Domains via Requirements to Software

- 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: Montanari Festschrift (2008); PSI'09 (2009)].

- 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.

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 talk is based on the engineering and study of many descriptions, of
  - air traffic,
  - banking,
  - commerce[2],
  - container lines,
  - health care,
  - logistics,
  - pipelines,
  - railway systems,
  - secure [IT] systems,
  - stock exchanges,

  etcetera.

[2] the consumer/retailer/wholesaler/producer supply chain

1.4. Contributions of This Talk

- 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` program and to conjecture the reverse; and, related to (ii),

(iii) suggesting complementing **syntactic** and **semantic** theories of mereology.

1.5. Structure of This Talk

We briefly overview the structure of this contribution.

- First, on Slides 15–31, **we loosely characterise how we look at mereologies: "what they are to us !"**.
- Then, on Slides 32–55, **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. 5.) focuses on the **syntax of mereologies.**

- Following that (Slides 56–69), **we indicate how the model of** the previous section **satisfies the axiom system of that section.**
- In preparation for the next section Slides 70–92 **presents characterisations of attributes of parts, whether atomic or composite.**
- Finally Slides 93–102 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.
- Lastly, Slides 103–106, concludes with some remarks on what we have achieved.

2. Our Concept of Mereology

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 neigbouring 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 mereological notion of **subpart**, that is: *contained within* can be illustrated by **examples:**
  - *the intersections and street segments are subparts of the road net;*
  - *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. 4 on Slide 23, as well as its aircrafts as adjacent within the air traffic system;*
  - *the pipes, valves, forks, joins and pumps of a pipeline, cf. Fig. 9 on Slide 28, as adjacent within the pipeline system;*
  - *two or more banks of a banking system, cf. Fig. 6 on Slide 25, 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".*

2.2. Six Examples

- We shall later
  - present a model that is claimed to abstract essential mereological properties of
    * air traffic,
    * buildings with installations,
    * machine assemblies,
    * financial service industry,
    * the oil industry and oil pipelines, and
    * railway nets.

2.2.1. Air Traffic

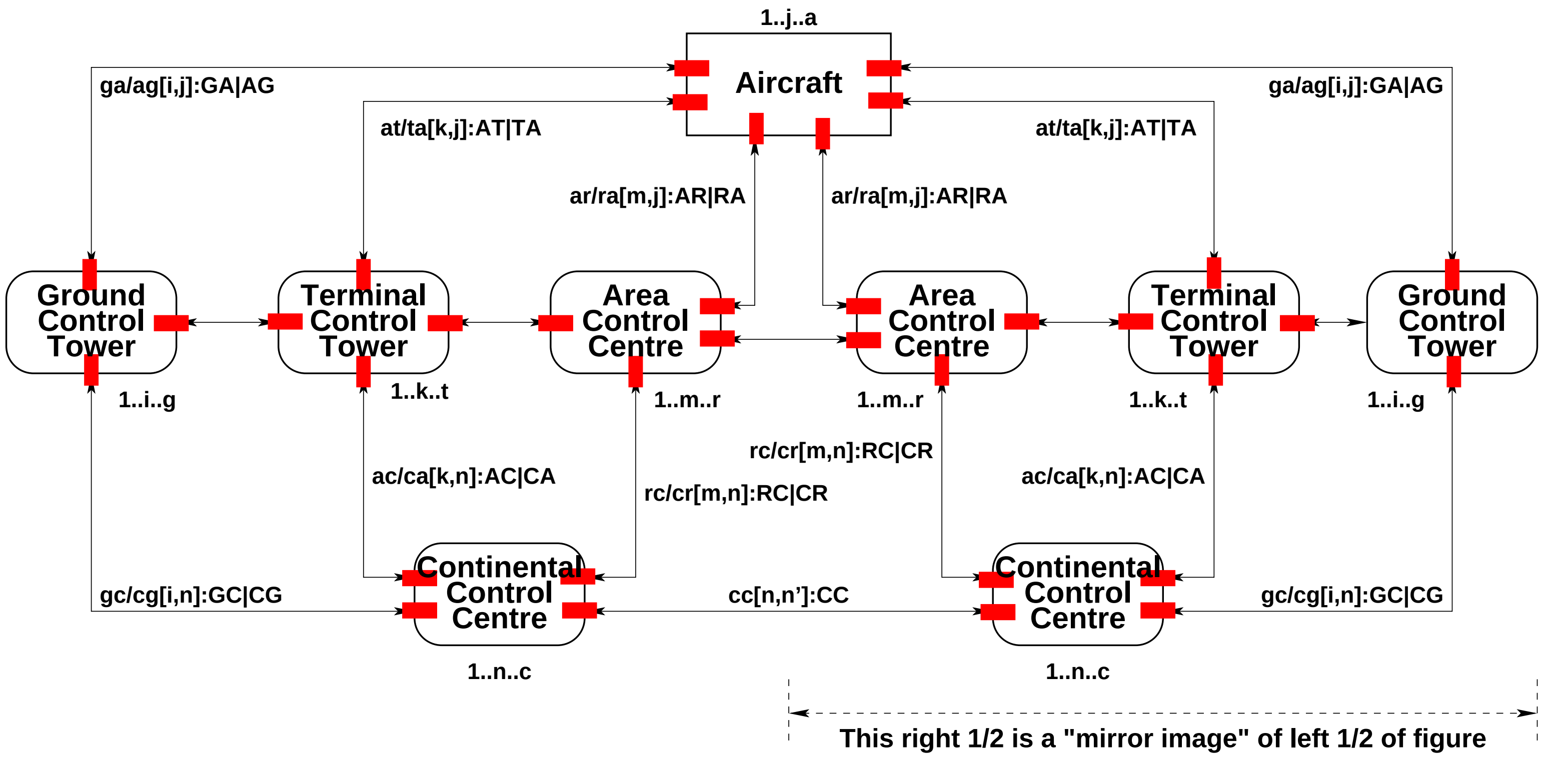

Figure 4: A schematic air traffic system

2.2.2. Buildings

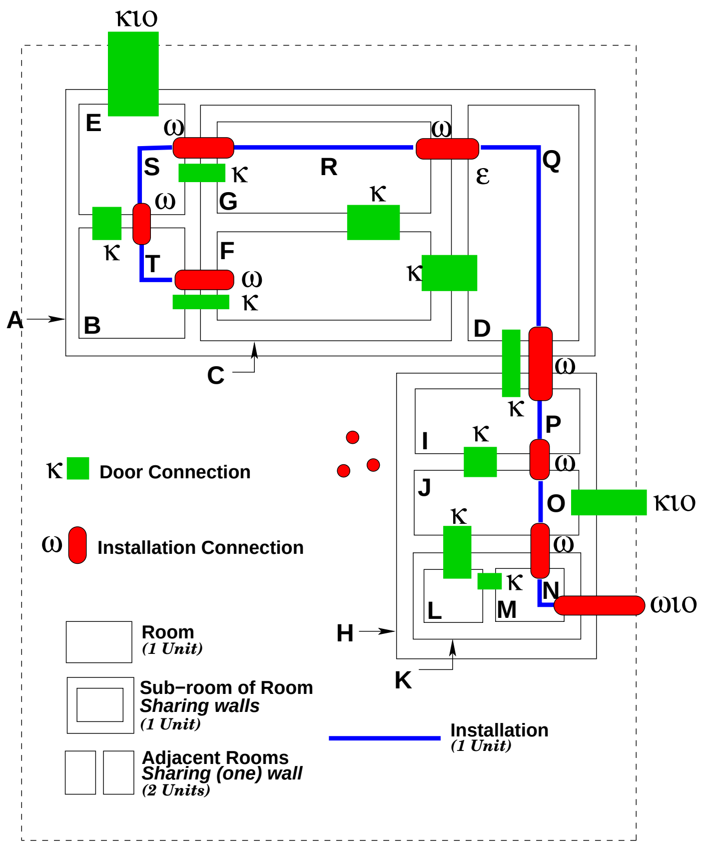

Figure 5: A building plan with installation

2.2.3. Financial Service Industry

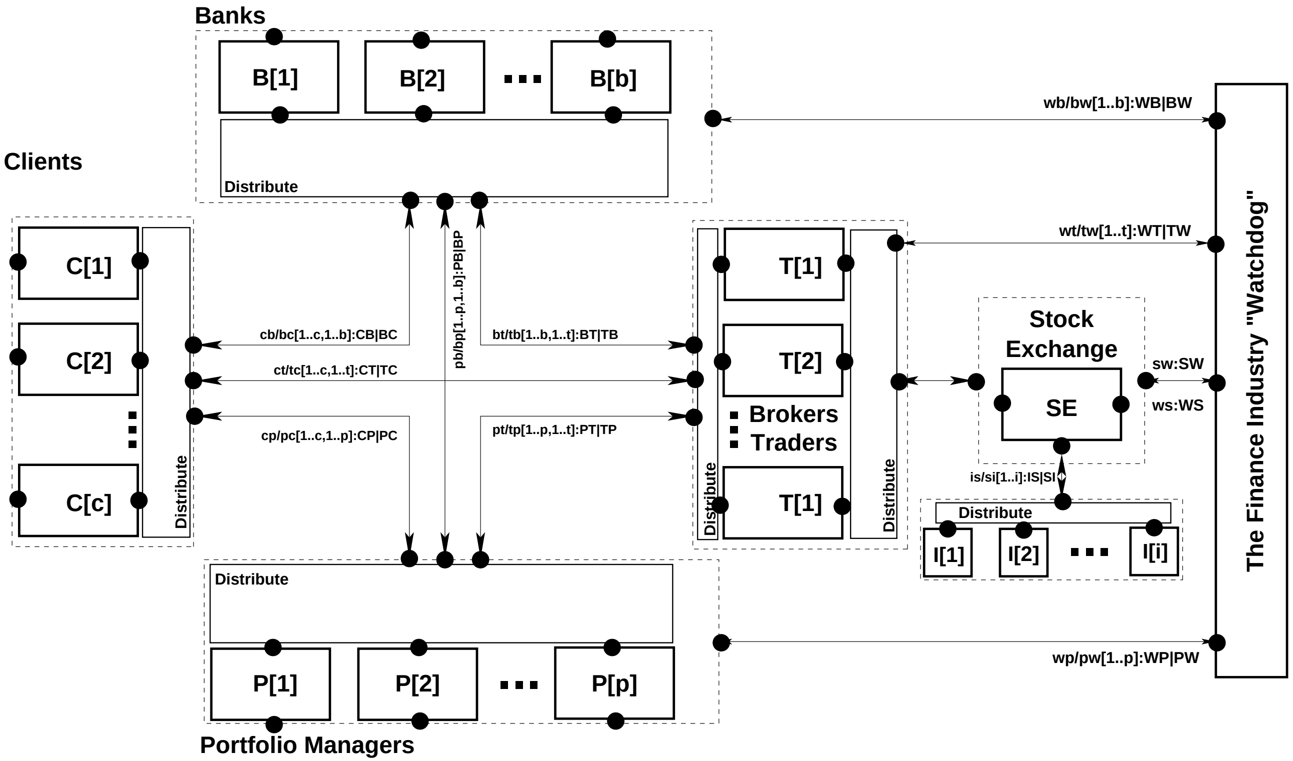

Figure 6: A financial service industry

2.2.4. Machine Assemblies

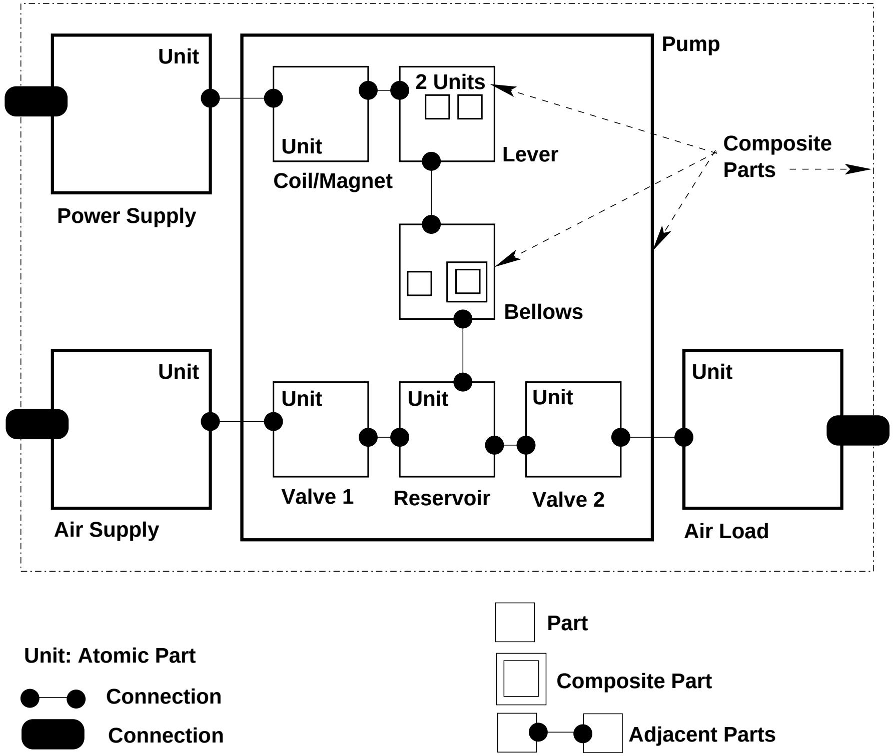

Figure 7: An air pump, i.e., a physical mechanical system

2.2.5. Oil Industry

2.2.5.1. "The" Overall Assembly

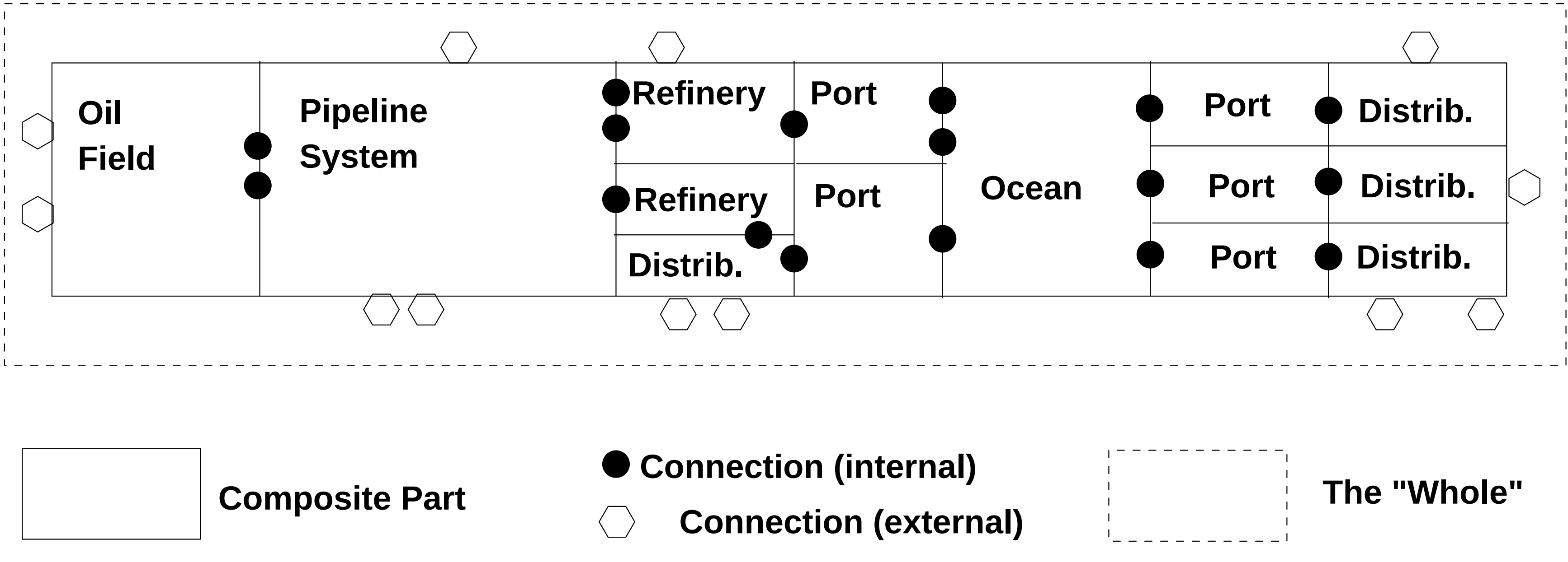

Figure 8: A Schematic of an Oil Industry

2.2.5.2. A Concretised Composite parts

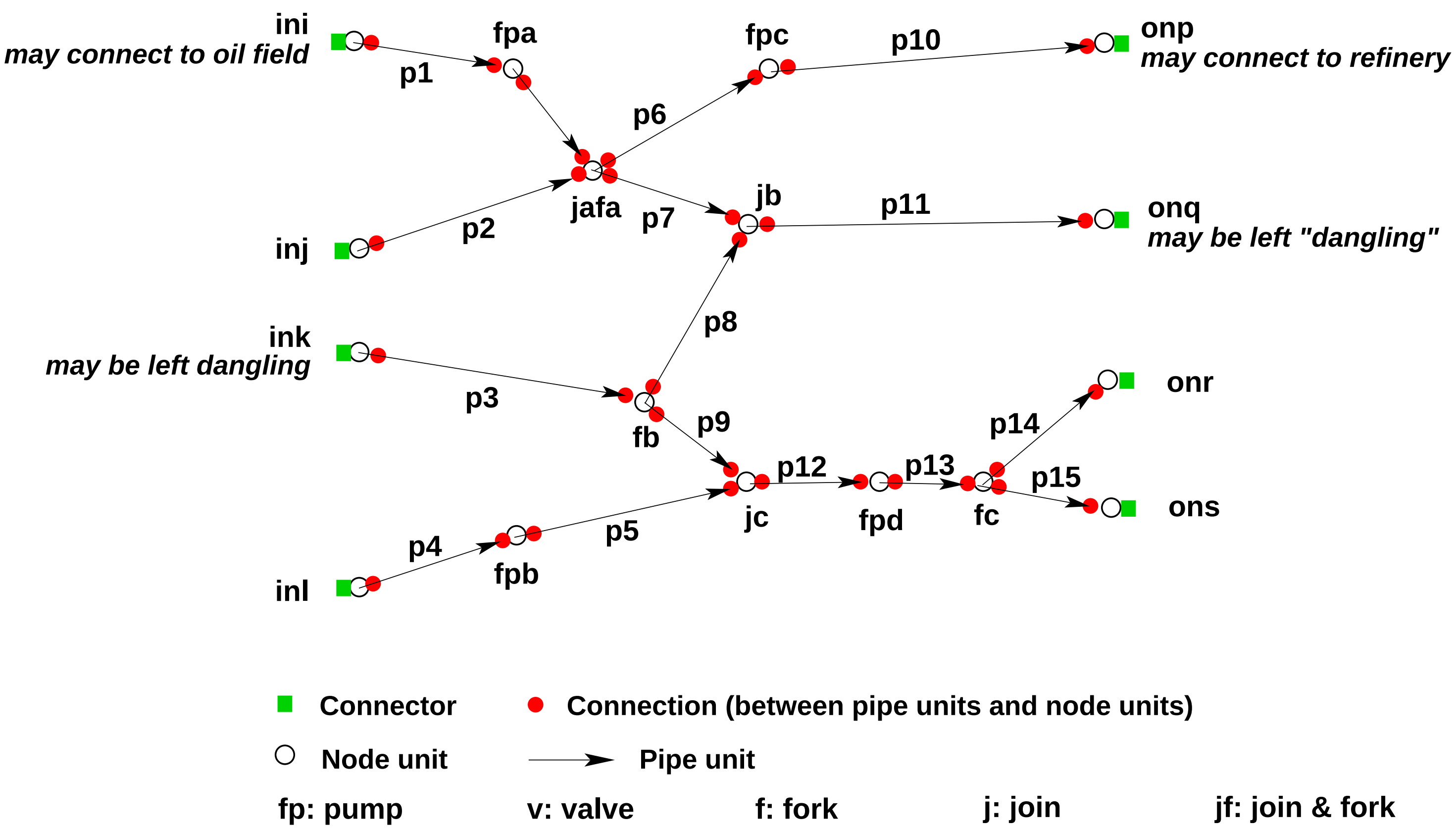

Figure 9: A pipeline system

2.2.6. Railway Nets

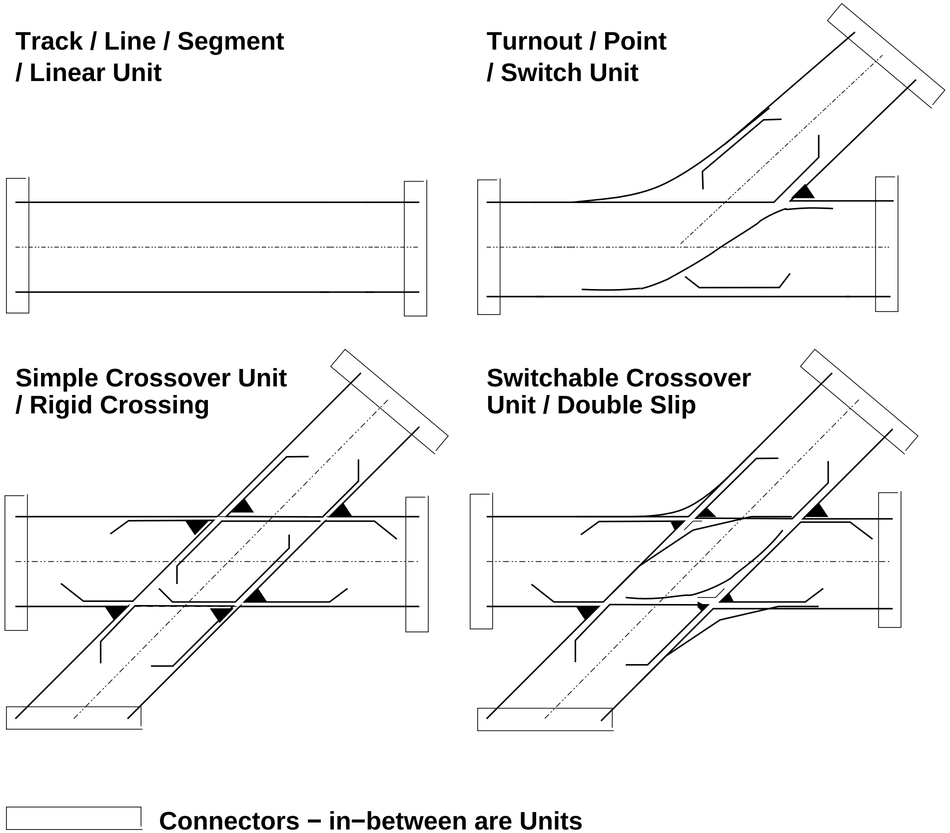

Figure 10: Four example rail units

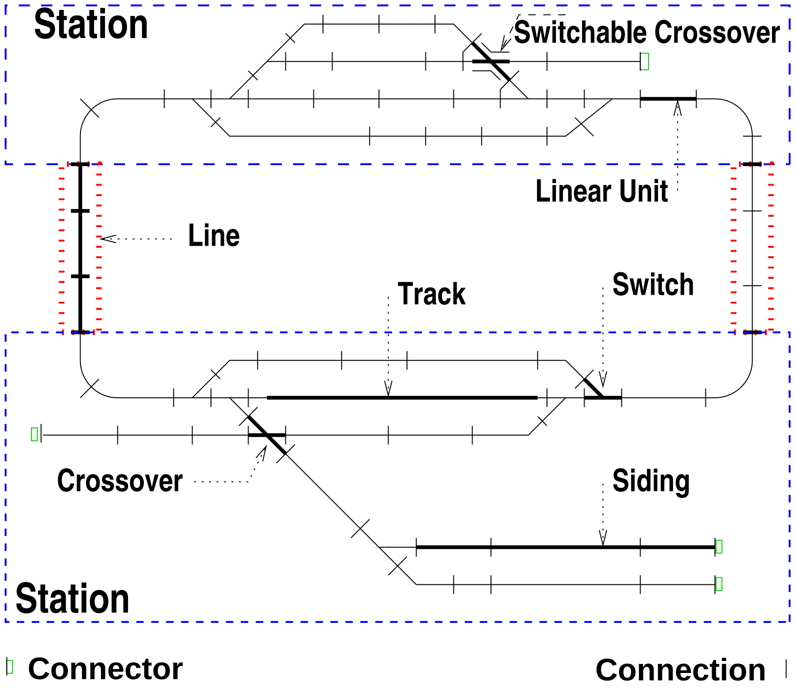

Figure 11: A "model" railway net. An Assembly of four Assemblies:
Two stations and two lines; Lines here consist of linear rail units;
stations of all the kinds of units shown in Fig. 10 on the preceding slide.
There are 66 connections and four "dangling" connectors

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.

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.

3.1. Parts and Subparts

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

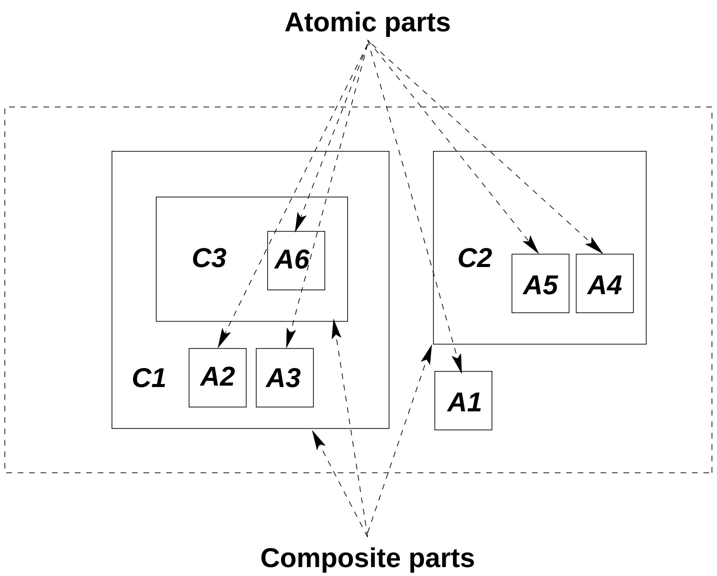

Figure 12: Atomic and composite parts

3.1.1. The Model

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.
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)≠{}

- Fig. 13 and the expressions below illustrate the observer function **obs_Ps**:
  - obs_Ps($C1$) = {$C2$, $C3$, $A1$},
  - obs_Ps($C2$) = {$A3$, $A4$},
  - obs_Ps($C3$) = {$A6$}.

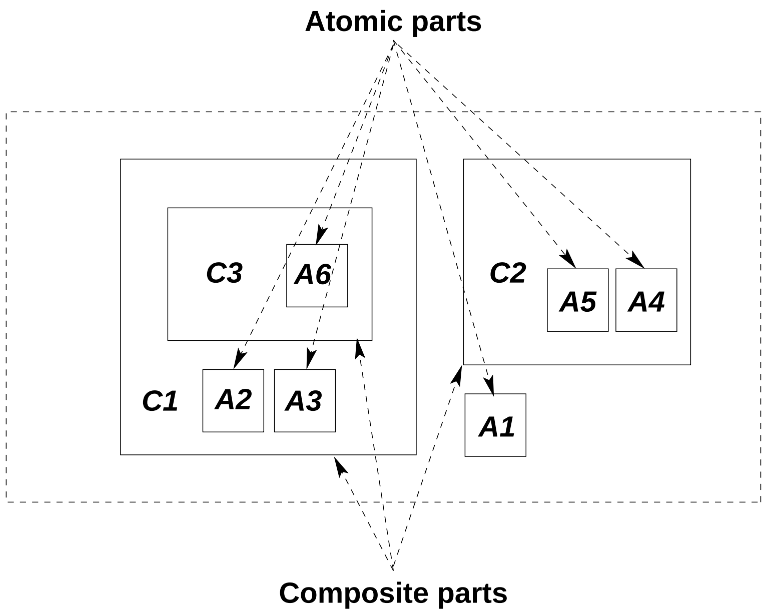

Figure 13: Atomic and composite parts

- 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. $\text{xtr\_Ps}: \text{C} \rightarrow \text{P-}\mathbf{set}$
6. $\text{xtr\_Ps}(\text{c}) \equiv$
6(a). $\quad \mathbf{let}\ \text{ps} = \text{obs\_Ps}(\text{c})\ \mathbf{in}$
6(b). $\quad \text{ps} \cup \cup \{\text{obs\_Ps}(\text{c}')|\text{c}'{:}\text{C} \cdot \text{c}' \in \text{ps}\}\ \mathbf{end}$

3.2. 'Within' and 'Adjacency' Relations

3.2.1. 'Within'

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

value

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

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

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: $P \times P \to C \xrightarrow{\sim}$ **Bool**,

9. imm_adjacent(p,p′)(c) $\equiv$ p$\neq$p′ $\wedge$ {p,p′}$\subseteq$obs_Ps(c)

3.2.4. Transitive 'Adjacency'

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 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 × P → C $\stackrel{\sim}{\rightarrow}$ **Bool**

10. adjacent(p,p′)(c) ≡

10(a). imm_adjacent(p,p′)(c) ∨

10(b). ∃ p′′,p′′′:P •

10((b))i. imm_adjacent(p′′,p′′′)(c) ∧

10((b))ii. ((p=p′′)∨within(p,p′′)(c)) ∧ ((p′=p′′′)∨within(p′,p′′′)(c))

3.3. Unique Identifications

- Each physical part can be uniquely distinguished
  - for example by an abstraction of its spatial location.
- 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 $\rightarrow$ Π
axiom
13. $\forall$ p,p′:P $\cdot$ p$\neq$p′ $\Rightarrow$ uid_Π(p)$\neq$uid_Π(p′)

- Figure 14 illustrates the unique identifications of composite and atomic parts.

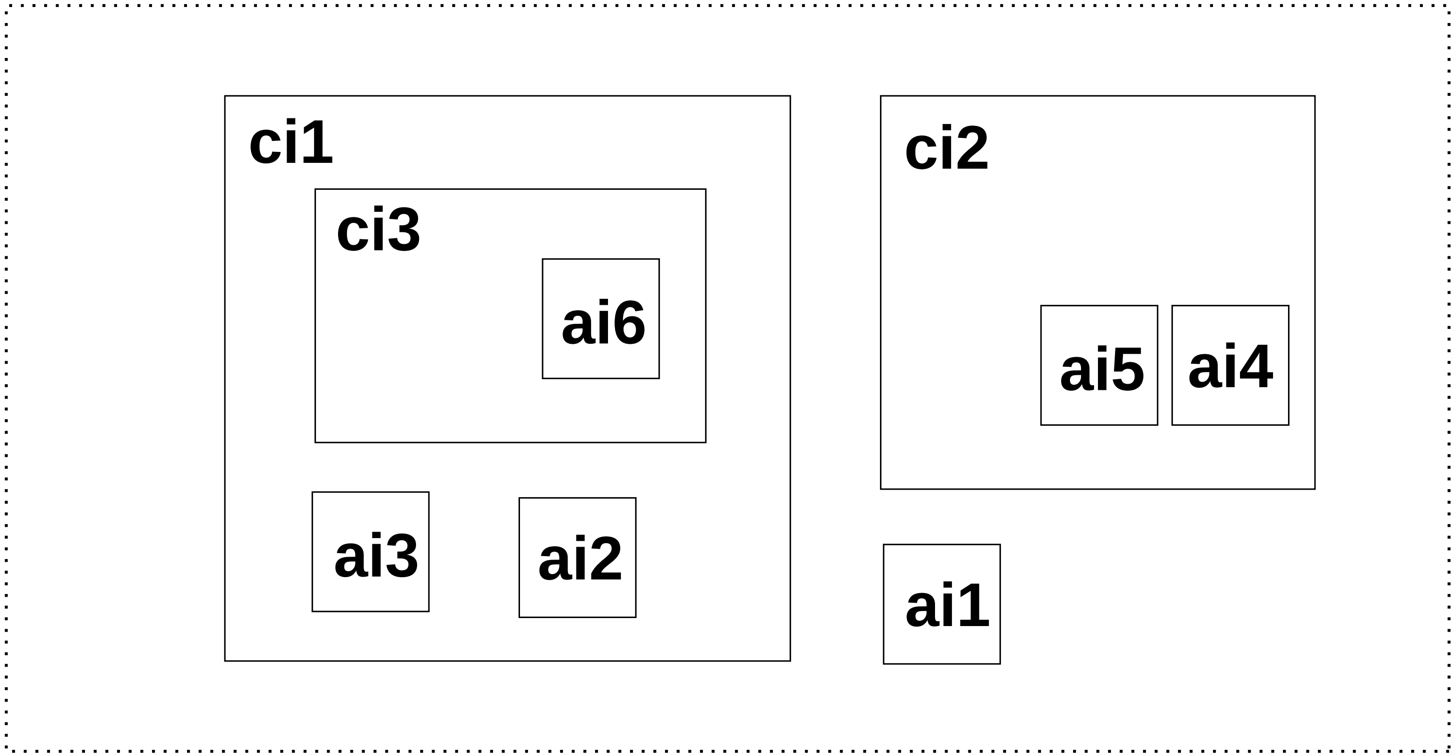

Figure 14: ai_j: atomic part identifiers, ci_k: composite part identifiers

- We exemplify the observer function obs_Π in the expressions below and on Fig. 15:
  - obs_Π(*C1*) = *ci1*, obs_Π(*C2*) = *ci2*, etcetera; and
  - obs_Π(*A1*) = *ai1*, obs_Π(*A2*) = *ai2*, etcetera.

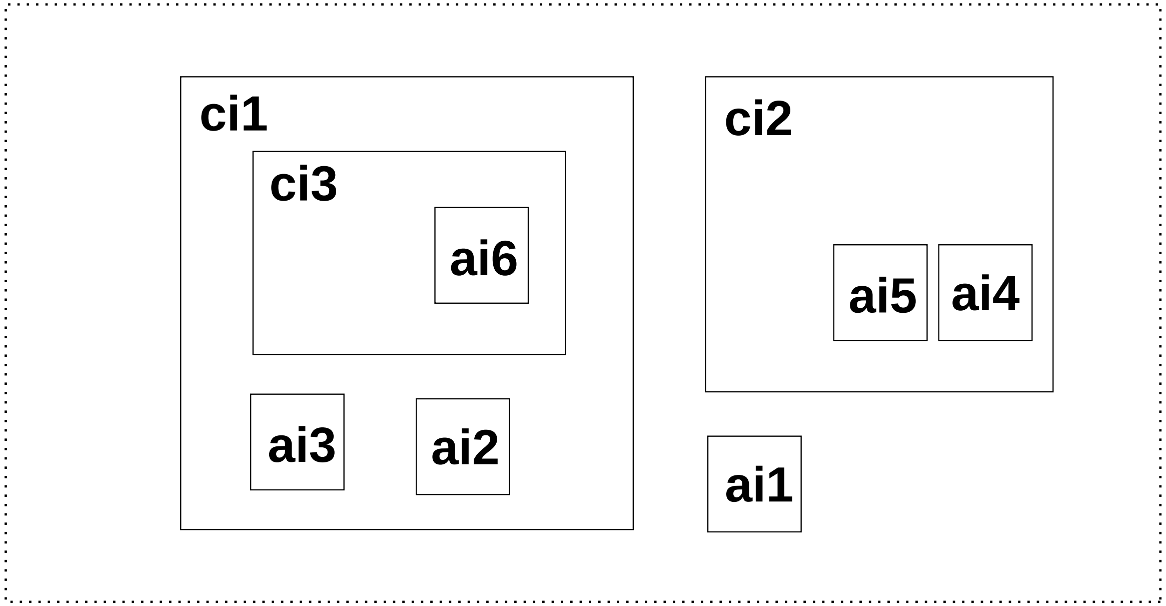

Figure 15: ai_j: atomic part identifiers, ci_k: composite part identifiers

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 → Π-**set**

14. xtr_Πs(c) ≡ {uid_Π(c)} ∪ {uid_Π(p)|p:P•p ∈ xtr_Πs(c)}

3.4. Attributes

- We shall later
  - 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**

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 16 on the facing slide illustrates connections between parts.
- A connector is, visually, a •—• line that connects two distinct part boxes.

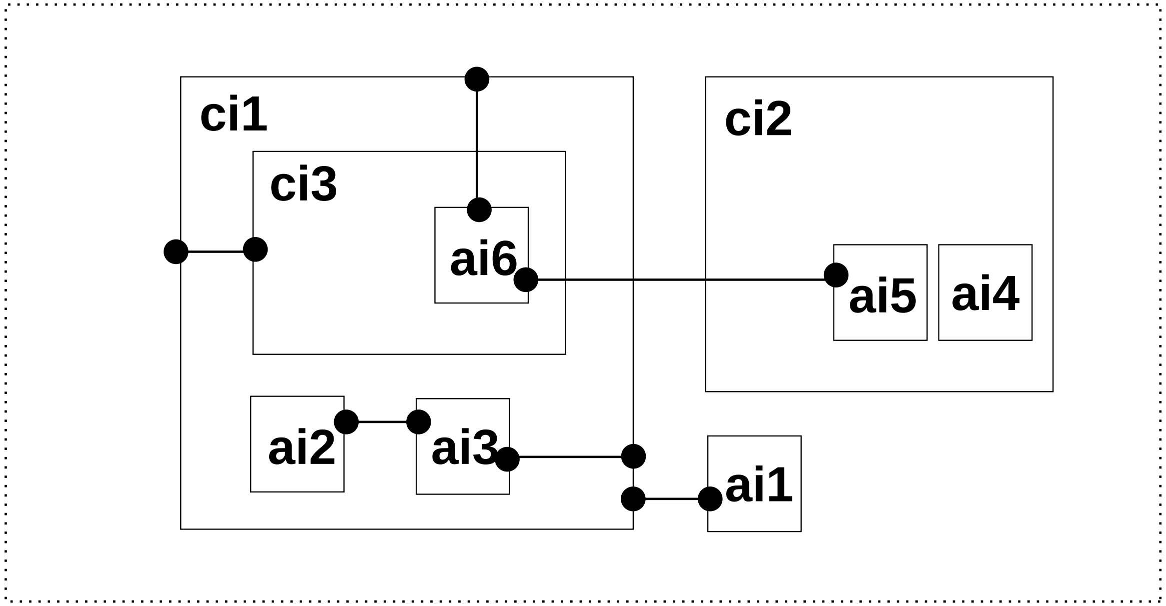

Figure 16: Connectors

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

For **example:**

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

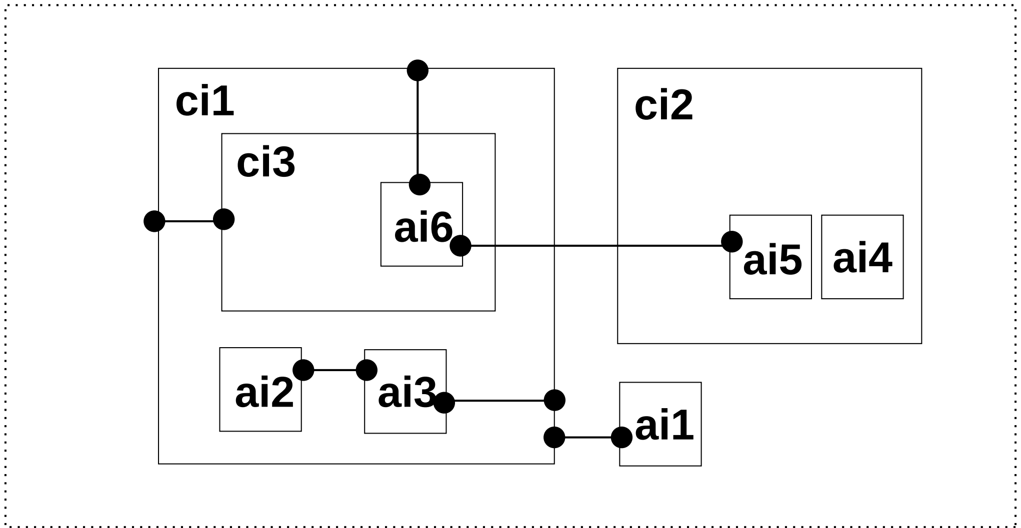

Figure 17: Connectors

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)}

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

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 Slide 49) can be extended to apply to **W**holes.

axiom

25. $\forall$ w:W•

25. **let** ps = xtr_Ps(w), ks = xtr_Ks(w) **in**

25(a). $\forall$ p,p′:P • p$\neq$p′ $\wedge$ {p,p′}$\subseteq$ps $\wedge$ share(p,p′) $\Rightarrow$

25(a). {uid_Π(p),uid_Π(p′)} $\in$ ks $\wedge$

25(b). $\forall$ {uid,uid′} $\in$ ks $\Rightarrow$

25(b). $\exists$ p,p′:P • {p,p′}$\subseteq$ps $\wedge$ {uid,uid′}={uid_Π(p),uid_Π(p′)}

25(b). $\Rightarrow$ share(p,p′) **end**

value

26. xtr_Ks: W $\rightarrow$ K-**set**

26. xtr_Ks(w) $\equiv$ $\cup${xtr_Ks(p)|p:P•p $\in$ obs_Ps(p)}

3.5.3. Sharing

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

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

3.6. Uniqueness of Parts

- There is one property of the model of wholes: W, Item 1 on Slide 34, and hence the model of composite and atomic parts and their unique identifiers "spun off" from W (Item 2 [Slide 34] to Item 25(b) [Slide 51]).
  - and that is that any two parts as revealed in different, say adjacent parts are indeed unique,
  - where we — simplifying — define uniqueness sôlely by the uniqueness of their identifiers.

3.6.1. Uniqueness of Embedded and Adjacent Parts

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

value

29. uiids: C → **Bool**

29. uiids(c) ≡ ∀ 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: unique identifiers).

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**

4. An Axiom System

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

4.1. Parts and Attributes

- In our axiom system for mereology we shall avail ourselves of two sorts:
  - $\mathcal{P}$arts, and
  - $\mathcal{A}$ttributes.[3]
  - **type** $\mathcal{P}, \mathcal{A}$
- $\mathcal{A}$ttributes are associated with $\mathcal{P}$arts.
- We do not say very much about attributes:
  - We think of attributes of parts to form possibly empty sets.
  - So we postulate a primitive predicate, $\in$, relating $\mathcal{P}$arts and $\mathcal{A}$ttributes.
- $\in: \mathcal{A} \times \mathcal{P} \to \mathbf{Bool}$.

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

4.2. The Axioms

- The axiom system to be developed in this section is a variant of that in [CasatiVarzi1999].
- We introduce the following relations between parts:

part_of:	$\mathbb{P} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 59
proper_part_of:	$\mathbb{PP} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 60
overlap:	$\mathbb{O} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 61
underlap:	$\mathbb{U} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 62
over_crossing:	$\mathbb{OX} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 63
under_crossing:	$\mathbb{UX} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 64
proper_overlap:	$\mathbb{PO} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 65
proper_underlap:	$\mathbb{PU} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 66

- Let $\mathbb{P}$ denote **part-hood**; p_x is part of p_y, is then expressed as $\mathbb{P}(p_x, p_y)$.[4]
  - (1) Part p_x is part of itself (reflexivity).
  - (2) If a part p_x is part p_y and, vice versa, part p_y is part of p_x, then $p_x = p_y$ (antisymmetry).
  - (3) If a part p_x is part of p_y and part p_y is part of p_z, then p_x is part of p_z (transitivity).

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

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

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

[4] 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.

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

- **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.

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

- **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.

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

- 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.

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

- **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.

$$\mathbb{UX}(p_x, p_y) \stackrel{\triangle}{=} \mathbb{U}(p_x, p_z) \land \neg\mathbb{P}(p_y, p_x) \tag{8}$$

- **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.

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

- **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.

$$\mathbb{PU}(p_x, p_y) \stackrel{\triangle}{=} \mathbb{UX}(p_x, p_y) \wedge \mathbb{UX}(p_y, p_x) \quad (10)$$

4.3. Satisfaction

- We shall sketch a proof that
  - the *model* of the previous section
  - *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.

4.3.1. A Proof Sketch

We assign

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

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

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 we now take a closer look at the concept of 'attributes'.
  - We consider three kinds of attributes:
    * their unique identifications [uid_Π]
      — which we have already considered;
    * their connections, i.e., their mereology [mereo_P]
      — which we also considered;
    * and their "other" attributes
      which we shall refer to as properties. [prop_P]

5.1. Mereological Properties

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

•

5.1.2. Unique Identifier and Mereology Types

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

type

  P = P1 | P2 | ... | Pn

  Π = Π1 | Π2 | ... | Πn

value

  uid_Πj: Pj → Πj for 1≤j≤n

  mereo_Πs: P → Π-**set**

- **Example:** Our example relates to the abstract model given earlier.

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 $\rightarrow$ Π

38. mereo_Πs: P $\rightarrow$ Π**-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** $(\pi,\pi s)$=(uid_Π,mereo_Πs)(p) **in**

39. $\{\{\pi',\pi''\}|\pi',\pi'':\Pi\cdot\pi'=\pi\wedge\pi''\in\pi s\}$ **end**

•

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 the next section.)

- Let us illustrate the concept of properties.
- **Examples:**
  - 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.

- Typical properties of street intersections are:
  * design[5]
  * location,
  * surface material,
  * surface condition,
  * traffic state — open or closed between any two pairs of in/out street segments.
- Typical properties of road nets are:
  * name,
  * owner,
  * public/private,
  * free/tool road,
  * area,
  * etcetera. •

[5] for example,

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

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

type
40. Props = PropNam $\overrightarrow{m}$ PropVAL
40(a). PropNam
40(b). 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)
40(d). xtr_PropVAL: P $\rightarrow$ PropNam $\tilde{\rightarrow}$ PropVAL
40(d). xtr_PropVAL(p)(pn) $\equiv$ (obs_Props(p))(pn)
40(d). **pre**: pn $\in$ xtr_PropNams(p)

- Here we leave PropNames and PropVALues undefined.

• Example:

type

NAME, OWNER, LEN, DESIGN, PP == public | private, ...

$L\Sigma$, $H\Sigma$, $L\Omega$, $H\Omega$

value

obs_Props: N → {| ["name"↦nm,"owner"↦ow,"public/private"↦pp,...]
| nm:NAME, ow:OWNER, ..., pp:PP |}

obs_Props: L → {| ["length"↦len,...,"state"↦lσ,"state space"↦lω:LΩ]
| **len**:LEN,...,lσ:LΣ,lω:LΩ |}

obs_Props: H → {| ["design"↦des, ...,"state"↦hσ,"state space"↦hω]
| des:DESIGN,...,hσ:HΣ,hω:HΩ |}

prop_NAME: N → NAME

prop_OWNER: N → OWNER

prop_LEN: L → LEN

prop_LΣ: L → LΣ, obs_LΩ: L → LΩ

prop_DESIGN: H → DESIGN

prop_HΣ: H → HΣ, obs_HΩ: H → HΩ

...

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 the section on ‘Attributes’ in the previous section, and to Items 15–16.

type

15.′ $\mathrm{ATR} = \Pi \times \Pi\text{-}\mathbf{set} \times \mathrm{Props}$

value

16.′ atr_ATR: $\mathrm{P} \to \mathrm{ATR}$

axiom

$\forall$ p:P $\cdot$ **let** $(\pi,\pi s,\mathrm{props})$ = atr_ATR(p) **in** $\pi \notin \pi s$ **end**

- In preparation for redefining the **share** function of Item 17 on Slide 45 we must first introduce a modification to property values.

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

type

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

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 × PropName → W → PropVAL
42. get_VAL(p,pn)(w) ≡
44. **let** pv = (obs_Props(p))(pn) **in**
42. **case** pv **of**
42. mk_Simp(v) → v,
42(a). mk_Π(π) →
42(a). **let** p′:P•p′ ∈ xtr_Ps(w)∧uid_Π(p′)=π **in**
42(c). (obs_Props(p′))(pn) **end**
42. **end end**
42(c). **pre**: pn ∈ obs_PropNams(p)
42(b). ∧ pn ∈ obs_PropNams(p′)
42(c). ∧ is_PropVAL((obs_Props(p′))(pn))

- The three bottom lines above, Items 42(b)–42(c), 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 $(\pi, \pi s,$ 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 $\rightarrow$ W $\rightarrow$ **Bool**
43(a). wf_ATR(π,πs,props)(w) $\equiv$
43(a). $\pi \notin \pi$s $\wedge$
43(b). $\forall$ π':Π $\cdot$ $\pi' \in$ **rng** props $\Rightarrow$
43(c). **let** pn:PropNam$\cdot$props(pn)=π' **in**
43(c). pi'$\in$ πs
43(d). $\wedge$ $\exists$ p':P$\cdot$p'$\in$ xtr_Ps(w)$\wedge$uid_Π(p')=π' $\Rightarrow$
43(e). pn $\in$ obs_PropNams(obs_Props(p'))
43(e). $\wedge$ $\exists$ mk_SimpVAL(v):VAL$\cdot$(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**
44(a). π ∈ πs′ ∨ π′ ∈ πs ∨
44(b). ∃ pn:PropNam•pn ∈ pns ∩ pns′ ⇒
44(b). **let** vop = props(pn), vop′ = props′(pn) **in**
44(b). **case** (vop,vop′) **of**
44(b). (mk_Π(π″),mk_Simp(v)) → π″=π′,
44(b). (mk_Simp(v),mk_Π(π″)) → π=π″,
44(b). _ → **false**
44. **end end end**

- **Comment:** v is a shared attribute.

5.4. Discussion

- We have now witnessed four kinds of observer function:
  - he above three kinds of mereology and property 'observers' and the
  - part (and subpart) **obs**_ervers,.
- These observer functions are postulated.
  - They cannot be defined.
  - They "just exist" by the force
    - of 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
  - we have not modelled distinctions between static and dynamic properties.
  - You may think, instead of such a model, that an **always** temporal operator, $\Box$, being applied to appropriate predicates.

6. A Semantic CSP Model of Mereology

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

6.1. A Semantic Model of a Class of Mereologies

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

6.1.1. Parts ≡ Processes

- The model of mereology (Slides 32–55) given earlier focused on (i) parts and (ii) connectors.
- To parts we associate CSP processes.
- Part processes are indexed by the unique part identifiers.
- The connectors form the mereological attributes of the model.

6.1.2. Connectors ≡ 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 on Slide 51) 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.

6.1.3. Process Definitions

value

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

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

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

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

ATR and atr_ATR are defined in Items 15.′ and 16.′ (Slide 83).

$$\text{atom\_process: a:A} \rightarrow \textbf{in},\textbf{out}\ \{\text{ch}[\,\text{cm(k)}\,]|\text{:K}\cdot\text{k} \in \text{cm(uid\_}\Pi\text{(a))}\}\ \textbf{process}$$
$$\text{atom\_process(a)} \equiv \mathcal{M}_{\mathcal{A}}\text{(a)(atr\_ATR(a))}$$

$$\mathcal{M}_{\mathcal{A}}\text{: a:A} \rightarrow \text{ATR} \rightarrow \textbf{in},\textbf{out}\ \{\text{ch}[\,\text{cm(k)}\,]|\text{k:K}\cdot\text{k} \in \text{cm(uid\_}\Pi\text{(a))}\}\ \textbf{process}$$
$$\mathcal{M}_{\mathcal{A}}\text{(a)(a\_attrs)} \equiv \mathcal{M}_{\mathcal{A}}\text{(a)(}A\mathcal{F}\text{(a)(a\_attrs))} \quad \textbf{assert:}\ \text{atr\_ATR(a)} \equiv \text{a\_attr}$$

$$A\mathcal{F}\text{: a:A} \rightarrow \text{ATR} \rightarrow \textbf{in},\textbf{out}\ \{\text{ch}[\,\text{em(k)}\,]|\text{k:K} \cdot \text{k} \in \text{cm(uid\_}\Pi\text{(a))}\}\ \ \text{ATR}$$

- The meaning processes $\mathcal{M}_{\mathcal{C}}$ and $\mathcal{M}_{\mathcal{A}}$ are generic.
  - Their sôle purpose is to provide a never ending recursion.
  - "In-between" they "make use" of Composite, respectively Atomic specific $\mathcal{F}$unctions
  - here symbolised by $C\mathcal{F}$, respectively $A\mathcal{F}$.
- Both $C\mathcal{F}$ and $A\mathcal{F}$
  - are expected to contain input/output clauses referencing the channels of their signatures;
  - these clauses enable the sharing of attributes.
- We illustrate this "sharing" by the schematised function $\mathcal{F}$ standing for either $C\mathcal{F}$ or $A\mathcal{F}$.

value

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

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

6.2. Discussion

6.2.1. General

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

6.2.2. Partial Evaluation

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

7. Closing

7.1. Relation to Other Work

- Douglas T. Ross: Plex, CAD, APT, SADT, IDEF0, . . .
- Leonard Goodman 1940: Calculus of Individuals
- R. Casati and A. Varzi: Parts and Places: the structures of spatial representation.
- B. Ganter and R. Wille: Formal Concept Analysis — Mathematical Foundations.
- Etcetera.

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 process (CSP).

7.3. Future Work

- We need to characterise, in a proper way,
  - the class of `CSP` programs
  - for which there corresponds a mereology.
- Are you game ?
- One could also wish for an extensive editing and publication of Doug Ross' surviving notes.

7.4. Acknowledgements

- I thank
  - Dr. Claudio Calosi and
  - Dr. Pierluigi Graziani,

  University of Urbino, Italy, for inviting this paper for
  - a Springer Verlag *Synthese Library* volume on
  - *Mereology and the Sciences*. Due Summer 2012
- I further thank Patricia M. Ross for permission to dedicate this paper to the memory of her husband of many years.

QUESTIONS ?