From rough to final designs by incremental set-inclusion of properties

Eir, A.; Ekholm, Anders

2002

From rough to final designs by incremental set–inclusion of properties

A. Eir
Department of Civil Engineering, and Informatics and Mathematical Modelling, DTU, Copenhagen, Denmark

A. Ekholm
Design Methodology, Department of Construction and Architecture, Lund Institute of Technology, Lund, Sweden

ABSTRACT: Design of buildings is a complex task in which ideas are sketched and communicated, by representations that are incrementally elaborated from the early rough sketches to the final design. We claim, that today's model–based design tools are restricted from fully supporting this process as they are founded on the principle that objects are instances of static types. Such systems do not offer work with objects being incrementally specialised according to their properties, and neither do they offer dynamics of the underlying type system.
The present paper elaborates on a property–oriented approach as a foundation for design tools facilitating incremental design based on set–inclusion of properties. We emphasize the formal foundation for incorporating such dynamics, and we specify requirements for tools facilitating incremental design and offering improved semantic support.

1 INTRODUCTION

The introduction of computer–aided design (CAD) using model–based tools has revolutionized many industries such as the automobile industry, the computer hardware industry, and the building industry. However, in building design the notion of creativity has suffered in favour of efficiency. One reason may be that most model–based design tools do not support incremental design; i.e. a process in which objects of the current model are being incrementally specialised according to their kinds.

In civil engineering and architecture, design is a creative process in which ideas are sketched and communicated, by representations that are incrementally elaborated from early rough sketches to the final design. In order for a model–based design tool to support (or at least not obstruct) creativity in this way, it should facilitate dynamics on four levels: (i) on the parts of the building, (ii) on the properties of parts, (iii) on other binary relations between parts; and, in addition, (iv) on the underlying type system. Most design tools facilitate the first, whereas they lack of functionality facilitating the three succeeding.

In this paper, we elaborate on a property–oriented approach as a foundation for design tools facilitating incremental specialisation of designs, based on set–inclusion of properties. We do so by offering a framework for dynamic management of objects and properties. This framework is well–founded in mathematical and computer science theories and disciplines like Formal Concept Analysis, Lattices, and Galois Connections. Furthermore, it strongly relates to formal methods and domain modelling as research disciplines of computer science (Bjørner 2000; Bjørner 1997). We sketch requirements for design tools facilitating incremental design and offering improved semantic support.

1.1 *The conceptual level*

Traditionally, CAD tools were made to create artefact descriptions like drawings, and for speeding up the design process by ensuring consistency, facilitating reuse, etc. With the introduction of 3D modelling and visualisation, graphical presentations like drawings (in a wide but still syntactical sense) are more important than ever.

However, there is more to a design process than the graphical entities of some presentation, as such a presentation can be seen as syntax (i.e. meaningful sequence of symbols) that certainly stands for something and is made according to some idea of an artefact. Cognitively, we understand artefact descriptions

In Eds Turk, Z. and Scherer R. "eWork and eBusiness in Architecture, Engineering and Construction". Proceedings of the European Conference on Product and Process Modelling, ECPPM 2002, Portoroz, Slovenia, Sep 9-11, pp. 293-300. Swets & Zeitlinger Publishers 2002.

like drawings as *presentations* of some conceptual model. Computer scientifically, we would say that artefact descriptions are computerized interpretations or evaluations of the conceptual model. Such a model is present in all stages of any design process and necessarily precedes any presentation-aimed syntax.

In order to develop design tools, we need to establish a conceptual understanding of the process of design and of design ideas.

1.2 *Survey*

Design as an incremental process is not a new concept. As an opposition to design tools focusing on the late stages of design, the Swedish BAS•CAAD project was initiated (Ekholm and Fridqvist 1995; Ekholm and Fridqvist 1998; Fridqvist 2000; van Leeuwen 1999), suggesting a computerized tool supporting design in the early stages. Also, the notion of *schema evolution* has been emphasized in context of design. For example, Hakim and Garrett propose an *Object–Centered* approach to modelling as a contrast to *Class–Centered* modelling, aiming the same as we, but using an informal reference frame; (Hakim and Garrett 1994; Garrett and Hakim 1994).

1.3 *Suggested framework*

In this paper, we elaborate on the *property–oriented* approach presented in the BAS•CAAD project. We see design as a process of exploration which includes choosing among alternatives, and incrementally adding parts, properties of parts, and other relations between parts. In the work mentioned in this paper, we focus the discussion on incremental property determination. What separates it from other work on incremental design is, that we intend to put on a logico–computer science perspective. Thereby, we go back and investigate the formal foundations for CAD. Especially, we focus on the notion of multiple inheritance of properties as a fundamental issue in design.

We claim that the (apparently) natural process of going from rough sketches to more and more precise designs is not entirely supported by today's commercial CAD tools. Rather, such tools emphasize the facility of incrementally adding objects (parts) of pre–defined static types, thereby focussing on partonomic (*part–whole*) relations (Artale, Franconi, Guarino, and Pazzi 1996) and not on taxonomic (*kind–of*) relations (Guarino and Welty 2000). The latter sort, we claim is as important to design as the former.

We agree with Turk et al (Turk 1998), that static type structures in the object–oriented paradigm restrict the practitioner and imply some sort of blindness. Though, we seek to approach the problem without issuing it as for or against *hermeneutic constructivism*. In our approach, we could say that incrementally stating the properties of objects in a design model, makes it possible for the designer to reflect on the design. Actually, making local, temporary restrictions like determining properties and even constraints, certainly offer a foundation for deciding how the design should be — or should not be. Total blindness in a design process may appear if we along the way keep too many open ends — we may not get anything done at all.

The framework, suggested here, sees properties as means for describing objects, but existing on the dynamic run–time level instead of the static. Thereby, we obtain the freedom mentioned above.

We rely on formal theories like Formal Concept Analysis, Lattices, and Galois Connections from knowledge engineering (Ganter and Wille 1999; Haav 1997; Guénoche and van Mechelen 1993) to establish this framework. Furthermore, we use The RAISE Specification Language (RAISE Language Group 1992; RAISE Method Group 1995) to specify requirements for design tools facilitating incremental design and offering improved semantic support, thus being suitable for distributed design. The formulae in RAISE are mathematically precise descriptions of the presented ideas, and aim to add a computer science perspective. However, the formulae are supplementary in the sense that they are not crucial for the understanding of this paper. For simplicity, formal specification of so–called well–formedness has be omitted.

2 THE DESIGN PROCESS

Design may be seen as a problem solving process, similar to problem solving in everyday life or in science. Such a process starts by expressing a problem definition. In order to recognize a problem one usually need some sort of goal, but not necessarily knowledge of how to reach that goal. Thus, the notion of a problem can be defined as *lack of solution knowledge in relation to background knowledge and goal* (Bunge 1983). A goal can be defined as an intended state of a system; i.e. the properties of the system at a certain time. In design, a goal could be understood as a satisfactory behaviour of an artefact.

Stating the problem definition is followed first by synthesis, describing a hypothesis or tentative problem solution (technical solution), and then of analysis investigating the proposed solution. The synthesis question is: Which system has these properties? And the analysis question is the inverse: What properties does this system have? Synthesis may be regarded as starting from a functional view on the system, while analysis starts from a compositional view. The result of the analysis is added to the background knowledge

and may lead to a revision of the goal. During the design process, hypotheses and tests are made alternately, and the properties of the intended artifact are determined incrementally. The design cycle, by (Simon 1981) called the "*Generator-Test Cycle*", proceeds until a satisfactory solution has been reached. The problem solving process is illustrated in Figure 1.

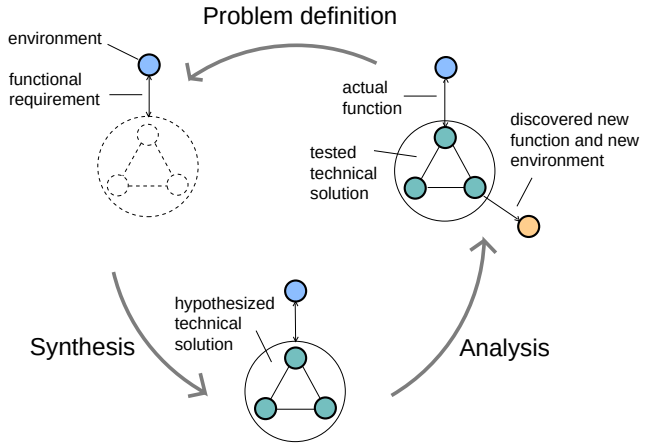

Figure 1: The problem solving process (Ekholm 2001).

2.1 *Incremental design: An example*

Consider the process of drawing a house to be built. Usually, the overall structure is sketched before large details like doors, windows, the sort of walls, etc. can be specified; although that may not always be the case.

Figure 2 shows three design stages of such a house: (a) a drawing of rough lines indicating the exterior structure, (b) a drawing of more precise geometry, doors added and wall type information, and (c) a drawing which in addition states the widths and lengths of walls.

We thus go from a rough sketch to more and more detailed specifications of the idea. By a rough sketch, we understand an artefact description which is *less* precise than a (relatively) more detailed specification.

Using predicate logic, we could formalize aspects of a wall object at two stages as:

$$\mathsf{wall}(x_1) \wedge \mathsf{straight}(x_1) \wedge \mathsf{masonry}(x_1) \quad (1)$$

$$\mathsf{wall}(x_1) \wedge \mathsf{straight}(x_1) \wedge \mathsf{masonry}(x_1) \wedge \mathsf{width}_{28}(x_1) \wedge \mathsf{length}_{568}(x_1) \quad (2)$$

We see, that Formula 1 and 2 are bound by implication from the latter to the former. That is, the former is a more rough description of the object x_1 than the latter.

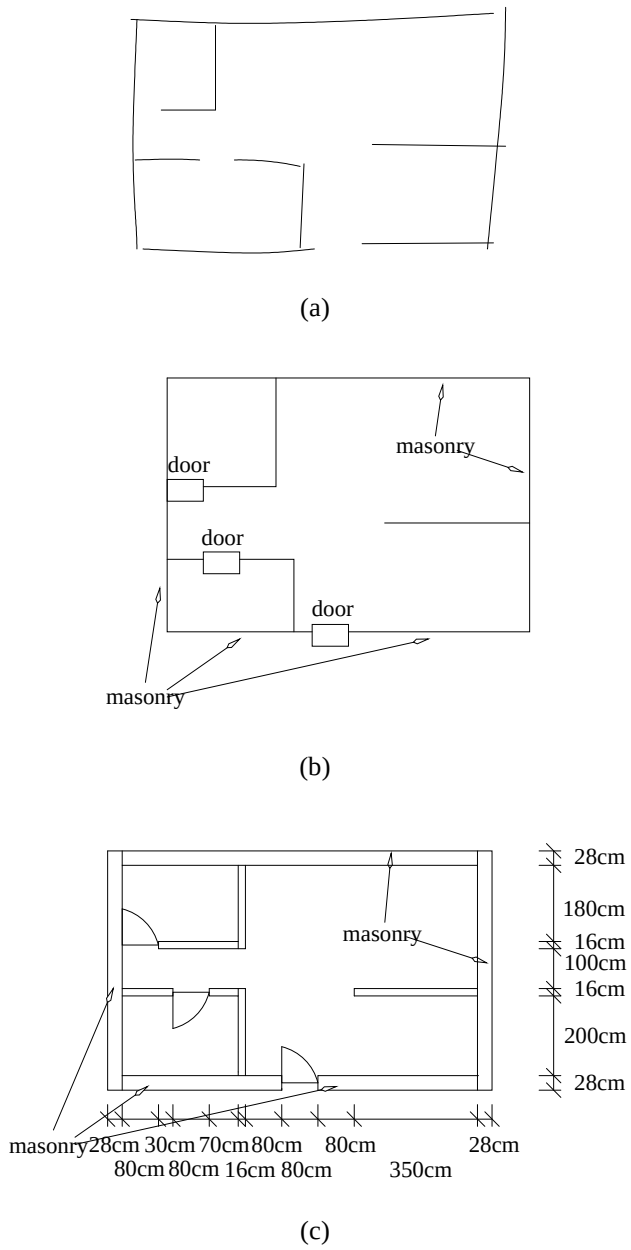

Figure 2: Rough sketch (a), simple drawing (b), more detailed drawing (c).

2.2 *Many–sorted knowledge and multiple inheritance*

An essential issue of design is that of adding different, many–sorted knowledge to an artefact description. That is, designing a building means adding different *incomparable* sorts of knowledge like material, colour, texture, etc. Furthermore, design is putting parts together on the conceptual level, thereby forming wholes, as well as adding other binary relations between objects.

Seen this way, design is: (i) multiple–inheritance as set–inclusion of Fregean (i.e. formalisable as unary predicates) properties (Frege 1994) like being red or made of wood, (ii) set–inclusion of parts like being composed by a number of bricks, and (iii) set–inclusion of other binary relations like for a wall to

approach and be connected to another wall.

Figure 3 shows how multiple inheritance of properties are meet (⊔) and design moves in separate incomparable directions origins from join (⊓) in a lattice structure (Birkhoff 1973).

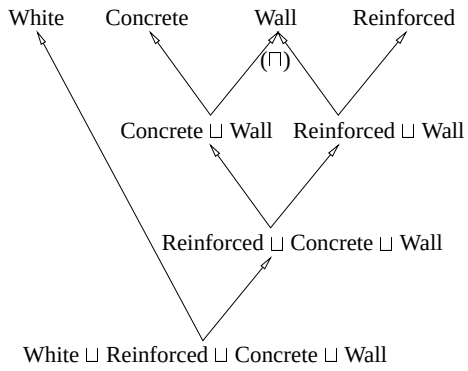

Figure 3: Design lattice.

3 TOOLS FOR DESIGN

In context of tools for design, the introduction of the object–oriented paradigm has *not* been more revolutionary than adding some convenient ways of working with data. That is, the real difference between classes and traditional data types is merely on some technicalities. In essence, types (even though classes) are still static.

The following two sections state the importance of abandoning this restriction by indicating requirements for design tools for incremental design.

3.1 *Dynamics on properties*

According to the object–oriented paradigm, objects are created as belonging to a certain class.

This has at least two consequences: (i) we cannot speak of loosely defined objects being incrementally specialised during run–time of the design tool, and (ii) objects cannot shift classes.

These facts, we claim, delimit today's design tools from fully supporting the creative process of design.

3.2 *Dynamics in type system*

A class in the object–oriented paradigm is a type in the language in which the tool is written. Thus adding a class definition or an attribute denoting a property, usually requires recompilation of the program code. This implies that classes and attributes cannot be changed or added after instantiation of any objects; (Hakim and Garrett 1994; Garrett and Hakim 1994).

In order to facilitate dynamics according to properties of objects and of the underlying type system, we need to move properties and relations of objects from the static type level to the dynamic run–time level (Ekholm and Fridqvist 1998). We do so by establishing a generic system in which properties and relations are values and thus can be managed at run–time. Furthermore, we have objects that are also values but mapped to their properties in an object environment store. The latter we shall call an *artefact model*.

This solution, however, has implications as we shall see in Section 6.

4 ARTEFACT MODEL

By an artefact model, we understand a configuration of objects, i.e. maps from conceptual objects to their properties and parts respectively, and a map from conceptual object pairs to relation designators. We found the specification of artefact models on the notion of *design moves* such that artefact models indeed facilitate incremental design.

4.1 *Design move*

The three stages of design from Section 2.1 are related by design moves. A design move is a transition from one artefact model a to another artefact model b; denoted $\Phi(a) \equiv b$. A design move $\Phi(a) \equiv b$ is valid, iff b is a description of an artefact idea which is a *relevant successor* (Galle 1999) to the artefact idea corresponding to the description b. Logically, the description b is a relevant successor of a, iff the artefact described taxonomically as well as partonomically and according to other binary relations, is a specialisation of the artefact described by a.

4.2 *Specification*

A formal specification of an artefact model should accomplish incremental specialisation and we thus specify the notion as a record with three entries: (i) The taxonomical relations as a map (t–map) from objects (x:X) to sets of properties (p:P), (ii) the partonomic relations as a map (p–map) from objects to their object parts, and (iii) additional binary relations as a map (r–map) from pairs of objects to sets of relation designators (υ:Υ):

type
Θ :: t:X $\xrightarrow{m}$ P-**set** p:X $\xrightarrow{m}$ X-**set** r:(X × X) $\xrightarrow{m}$ Υ-**set**,
X, P, Υ

Note, that part-whole relations are singled out as a special case of binary relations between objects.

The wall modelled in Formula 2, could be represented by the following artefact model:

value

x_1: X, wall, straight, masonry, $width_{28}$, $length_{568}$:P,
θ:Θ = mk_Θ([$x_1 \mapsto$ {wall, straight, masonry, $width_{28}$, $length_{568}$}],
[$x_1 \mapsto$ {}],[ ])

Using the specification of Θ, we define the notion of design move as:

value
Φ: Θ → Θ,
≤: Θ → Θ → **Bool**

axiom ∀ θ, θ':Θ •
$\theta \leq \theta' \equiv$
let (t,p,r)=θ, (t′,p′,r′)=θ' **in**
(∀ x:X • x ∈ **dom** t ⇒ x ∈ **dom** t′ ∧ t(x) ⊆ t′(x)) ∧
(∀ x:X • x ∈ **dom** p ⇒ x ∈ **dom** p′ ∧ p(x) ⊆ p′(x)) ∧
(∀ (x,x′):X×X • (x,x′) ∈ **dom** r ⇒ (x,x′) ∈ **dom** r′ ∧
r(x,x′) ⊆ r′(x,x′))
end,
Φ(θ) **as** θ'
post $\theta \leq \theta'$

It can be shown that the relation ≤ defines a partial ordering on artefact models.

5 TOWARDS PROPERTY–ORIENTATION

The backbone of any ontology–based information system is the *kind–of* relation and the relation between objects and their properties. The foundation for these relations is the mathematical notion of a *Galois Connection* which can be said to hold between two power sets (Ganter and Wille 1999). A Galois Connection between sets of objects and sets of their common properties has the convenience that it defines a partial ordering of the properties. Furthermore, the partial ordering is a lattice which facilitates knowledge querying and management.

In the following sections, we give a short introduction to Galois Connections in order to justify the presented specification of artefact models and its focus on properties.

5.1 *Galois Connection*

We can say that an object x has the property y. For that we write xRy where R is called the *incidence relation*; (Ganter and Wille 1999). The correspondence between objects and their common properties becomes quite an important relation, as the functions giving the common properties of an object set and the residual function appears to be a Galois Connection.

Definition 1 *A pair of mappings (f, g), with respect to the power sets X-**set** and P-**set**, is a Galois Connection iff f and g are monotonously decreasing:*

type
X, P

value
f: X-**set** → *P*-**set**,
g: P-**set** → *X*-**set**

axiom
(∀ *xs,xs′:X*-**set** • *xs* ⊆ *xs′* ⇒ *f(xs′)* ⊆ *f(xs)*) ∧
(∀ *ps,ps′:P*-**set** • *ps* ⊆ *ps′* ⇒ *g(ps′)* ⊆ *g(ps)*) ∧
(∀ *xs:X*-**set** • *xs* ⊆ *g(f(xs)))* ∧
(∀ *ps:P*-**set** • *ps* ⊆ *f(g(ps)))*

The two mappings are then called *dually adjoint* (Ganter and Wille 1999). The above rather sloppy specification assumes the existence of only *one* incidence relation.
We now define f and g in the following way, as we model the incidence relation as a map (m:M) from object-property–pairs to boolean values:

type
M = (X × P) $\xrightarrow{m}$ **Bool**

axiom ∀ x:X,p:P,xs:X-**set**, ps:P-**set**, m:M •
f_m(xs) ≡ {p|p:P • (∀ x:X • x ∈ xs ∧ (x,p) ∈ **dom** m ⇒ m(x,p))},
g_m(ps) ≡ {x|x:X • (∀ p:P • p ∈ ps ∧ (x,p) ∈ **dom** m ⇒ m(x,p))}

Note, that we for f and g assume the presence of a formal context (m:M). The pair (f, g) satisfies the criteria for being a Galois Connection (Ganter and Wille 1999). This has the advantage that properties are partially ordered and forms a lattice structure, which can be utilized in semantic query and support facilities in design tools as sketched in Section 6.2.

The important taxonomical relation is modelled in Θ as the t–map. The relation is essential; a backbone in all ontological systems, and can be formulated as unary predicates, like being a wall ($\mathsf{wall}(x)$), or having a length of $4cm$ ($\mathsf{width\_4}(x)$). The partial ordering of properties is fundamental, as it e.g. places the property of being a door as a more general property than e.g. being a left–hand door, in the lattice.

The t–map, however, is not the only structure forming a lattice. Also the p–map and r–map form lattices because of their set–based nature.

In knowledge engineering, attempts have been made for unifying taxonomic and partonomic relations using the Pierce product as notation for attribution; (Brink, Britz, and Schmidt 1996). In Θ, the three maps can be seen as three different aspects (Simons 1987) of the artefact described.

5.2 *Relation to the BAS•CAAD ThingClass*

In this section, we relate the presented artefact model to the *ThingClass* of the BAS•CAAD project

(Ekholm and Fridqvist 1999), from which it was inspired. A ThingClass is defined as a 6–tuple of attribute sets, denoted: $T = (T_G, T_C, R_I, T_E, R_E, A_U)$. T_G is the set of generic or superclass attributes, T_C is the set of composition attributes, R_I is the set of internal relations, T_E is the set of environment attributes, R_E is the set of external relations, and A_U is the set of unary attributes which represent intrinsic properties of systems.

The specification of the presented artefact model Θ can be considered a projection of the above ThingClass specification.

The members in the union set of generic or superclass attributes (T_G) and unary attributes representing intrinsic properties (A_U) each designate properties (p:P) in Θ. We have not made a distinction between class attributes and intrinsic properties, as we formally cannot make this distinction. Both may be denoted by unary predicates, so the distinction is merely intentional.

The set of composition attributes, corresponds to the p–map in Θ.

The sets of environment attributes and external relations have been omitted in Θ. In the ThingClass these aim at modelling functional requirements (or behaviour) of an artefact, but such information might exist on a different level of description. Formally instantiating a binary relation between an artefact and the environment implies considering that environment piece as part of the artefact. Rather, it is combinations of properties and relations that makes the artefact functional. E.g. a certain combination of dimension and material makes a wall resist fire. Thus, functional requirements are to be modelled as a mathematical function ranging over sets of properties and relations. This way we are able to model what Bunge's calls *dispositional properties* (Bunge 1977).

A further study of this area could be founded on understanding properties as functions of events (Shoemaker 1997).

6 DESIGN TOOL REQUIREMENTS

In this section, we specify some basic requirements for design tools facilitating incremental design and improved semantic support. We found this specification on an organisation of information levels. We have already mentioned the conceptual level on which artefact descriptions belong. In addition, there are two more levels: The *presentation* level on which visualisations of artefact models belong, and the *semantic descriptive* level on which the semantical functions of properties, relations, design moves, etc. belong. The mathematical function modelling dispositional properties (see Section 5.2) appears as a special semantic function (see (Eir 2000) for another denotational semantic treatment of the domain of civil engineering).

Information on the presentation level is the result of computerized interpretation of information on the conceptual level, according to information on the semantic descriptive level.

Figure 4 shows the semantic relations between the three levels.

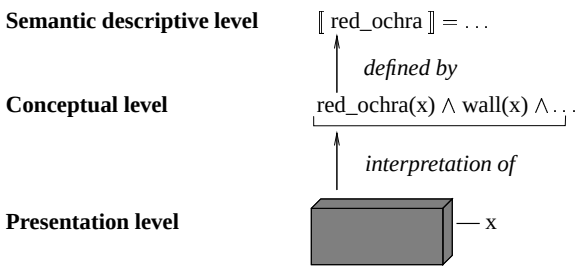

Figure 4: Information Levels.

Adding properties and various sorts of relations to an artefact model, we consider a special case of semantic descriptive information (although it certainly differs from the understanding of specifying the semantics of properties).

6.1 *Semantics of design moves*

The semantics of design moves can be described as functions from one artefact model to another. That is, the various sorts of moves like adding a part, a property to an object, etc. are considered syntactical commands (c:Cmd) on the semantic descriptive level. The semantics (specified using the semantical parentheses ⟦ ⟧) we model as the result/effect of performing the move; here using a denotational approach (Eir 2000):

type
Cmd == addobj | addprop(p:P) | addpartrel(x:X, x′:X) |
addrel(z:X, z′:X, υ:Υ),

value
⟦ ⟧: Cmd $\xrightarrow{\sim}$ Θ $\xrightarrow{\sim}$ Θ,
new: Θ → X

axiom ∀ c:Cmd, θ, θ':Θ, p:P, x,x′:X •
⟦ addobj ⟧(θ) ≡
let x=new θ **in**
mk_(t(θ)†[x ↦ {}],p(θ)†[x ↦ {}],r(θ))
end,

⟦ addprop(p) ⟧(θ) ≡
mk_Θ(t(θ)†[x ↦ t(θ) ∪ {p}],p(θ),r(θ)),

⟦ addpartrel(x, x') ⟧(θ) ≡
mk_Θ(t(θ),p(θ)†[x′ ↦ p(θ) ∪ {x}],r(θ)),

⟦ addrel(x, x', υ) ⟧(θ) ≡
mk_Θ(t(θ),p(θ),r(θ)†[(x,x′) ↦ r(θ) ∪ {υ}]),

6.2 *Semantic support*

The presented property–oriented framework facilitates a large number of semantic applications. Of these, we emphasize: Querying design models (e.g. as conformance checking), and merging artefact models. From an artefact model, we can extract the formal context (m:M) by determining the property–set (ps:P-**set**) present in the artefact model. We then build the map for the formal context by pairing objects (x:X) from the artefact model with each property in ps and mapping the pair to a boolean value (b:**Bool**) stating whether or not the object has this property. This context serves as the taxonomical information we need in simple querying.

value
X_M: Θ → M
X_M(θ) ≡
  let ps={p|p:P • (∃ x:X • x ∈ **dom** t(θ) ∧
   p ∈ t(θ)(x))} **in**
   [(x′,p′)↦b | x′:X,p′:P,b:**Bool** • x′ ∈ **dom** t(θ) ∧
     p ∈ ps ∧ b = p ∈ t(θ)(x)]
  end

A simple form for querying is one which returns the set of objects that satisfies certain criteria according to possessed properties and relations. That is, a query (q:Q) is a tuple of the form (P-**set** × (ϒ $\overrightarrow{m}$ X-**set**)). Leaving all X-**set** empty yields partially evaluation assuming some ϒ–relation to *any* object.

type
Q = P-**set** × (ϒ $\overrightarrow{m}$ X-**set**),

value
$\mathcal{I}$: Q → Θ → X-**set**
$\mathcal{I}$(ps,rel)(θ) ≡
  {x|x:X • (∀ p:P • p ∈ ps ⇒ p ∈ t(θ)(x)) ∧
  ∀ υ:ϒ • υ ∈ **dom** rel ⇒
    υ ∈ **dom** r(θ) ∧
  (∃ x′:X • x′ ∈ rel(υ) ⇒ υ ∈ r(θ)(x,x′))}

The above can be specialised such that interpretations of queries return (sub–)artefact models. This can be convenient when writing various extraction applications.

Furthermore, by specifying requirements as sets of properties and relations, it is possible to perform conformance checking on artefact models.

As a result of distributed design processes, e.g. Web–based, a function for merging two artefact models may be useful. The function should, given two artefact models, state what possible inconsistency there is. We state such as an artefact model itself. That is, the set of objects mapping to conflicting properties, and the objects making a cycle in the part–whole relations, see (RAISE Language Group 1992) for a formal definition. We assume, the existence of a predicate *conflict* stating whether or not two properties are mutually exclusive; based on some "universal" taxonomy represented as a formal context.

value
merge: Θ × Θ → M → Θ
merge(θ,θ′)(m) **as** θ″
post (∀ x:X • x ∈ **dom** t(θ″)) ⇒
  (∀ (p,p′):(P×P) • {p,p′} ⊆ t(θ″)(x) ⇒ conflict(p,p′)(m)) ∧
  p(θ″) = [x↦xs|x:X,xs:X-**set** •
   (x ∈ p(θ) ∧ x ∈ p(θ′) ⇒
     xs=p(θ)(x) ∪ p(θ′)(x)) ∨
   (x ∈ p(θ) ⇒ xs=p(θ)(x)) ∨
   (x ∈ p(θ′) ⇒ xs=p(θ′)(x))] ∧
   is_cyclic(p(θ″))),
  conflict: (P × P) → M → **Bool**,
  is_cyclic: (X $\overrightarrow{m}$ X-**set**) → **Bool**

Simple consistency of *one* artefact model (θ:Θ) can be obtained by using the empty artefact model as second argument. That is, merge(θ,([],[],[])) applied on some formal context, e.g. X_M(θ).

6.3 *Accumulating design knowledge*

Even though we have assumed that an artefact model only applies to one artefact idea, the notion has more potential. Consider the case of making a whole series of artefact models $\theta_1, \theta_2, \ldots, \theta_n$. These, we assume represent different artefact ideas, each on their "final" stages of design. The knowledge within these might be important for future design processes in two ways: (i) Each artefact model defines taxonomic relations, and (ii) each artefact model can be seen as examples of how to model certain artefacts. The two can be quite important in knowledge engineering of artefacts. It can be shown that formal contexts representing taxonomic relations easily can be added incrementally. Thereby, knowledge of *kind–of* relations can be built up and concepts involving various incomparable properties can be deduced; (Ganter and Wille 1999). Such concepts can then be made predefined types such that efficiency of design (like we have it in today's commercial CAD tools) is achieved. Furthermore, collecting previous artefact models together with the semantic definitions of e.g. properties, we have a (though primitive) foundation for defining concepts by their extentions. That is, simply by visualising previous objects falling under the concept in question.

7 CONCLUSION: BEYOND DRAWINGS

We have presented a property–oriented framework for design tools supporting incremental design. In order to do so, we have specified the notion of an artefact model storing taxonomic, partonomic and other binary relations of the artefact idea. An achievement is

that properties and relations can be added incrementally to the current artefact model. That is, we abandon the idea of static type systems and move properties and relations to the dynamic run–time level. Furthermore, we have indicated how new properties and relations can be introduced by formally specifying their semantics. We have argued that the aspects of any artefact model separately satisfies Galois Criteria, and thus makes the specification a solid foundation for tools offering improved semantic support. In this context, we have specified some basic requirements.

However, the framework can be taken further. In this paper, we have not really relied on artefact descriptions as syntactical documents like drawings. Rather, we have focused on the underlying conceptual framework, being the semantical one. A similar framework could certainly be defined for other sorts of civil engineering tools like for managing construction specifications or handling contracts. We believe that a wide range of civil engineering tools might benefit from a semantic treatment and thus can be founded on a property–oriented framework.

REFERENCES

Artale, A., E. Franconi, N. Guarino, and L. Pazzi (1996). Part-whole relations in object-centered systems: An overview. *Data & Knowledge Engineering 20*(3), 347–383.

Birkhoff, G. (1973). *Lattice Theory* (third ed.). Providence, R.I.: Amer. Math. Soc. Coll. Publ.

Bjørner, D. (1997, October). Domains as a prerequisite for requirements and software: Domain perspectives and facets, requirements aspects and software views. In *Proceedings US DoD/ONR Workshop, Bernried*.

Bjørner, D. (2000, august). Informatics: A truly interdisciplinary science – prospects for an emerging world. In *Proc. AIT, Bangkok, Thailand; invited paper*, pp. 71–84.

Brink, C., K. Britz, and R. A. Schmidt (1996). Peirce algebras. *Formal Aspects of Computing 6*, 339–358.

Bunge, M. (1977). *Ontology I: The Furniture of the World*, Volume 3 of *Treatise on Basic Philosophy*. Reidel, Dordrecht and Boston.

Bunge, M. (1983). *Epistemology and Methodology I: Exploring the World*, Volume 5 of *Treatise on Basic Philosophy*. Reidel, Dordrecht and Boston.

Eir, A. (2000). A semi–semantic document system for construction specifications. Master's thesis, Department of Information Technology, Technical University of Denmark.

Ekholm, A. (2001). Information systems for architectural design — Experiences from the BAS•CAAD and ACTIVITY projects. *Nordic Journal of Architectural Research 3*, 79–86.

Ekholm, A. and S. Fridqvist (1995, August). Object–oriented caad — design object structure and models for buildings, user organisation and site. In M. Fisher, K. Law, and B. Luiten (Eds.), *Modeling of buildings through their life cycle, CIB W78/TG10 workshop on computers and information in construction*, pp. 553–564. Stanford University, Ca, USA.

Ekholm, A. and S. Fridqvist (1998, June). A dynamic information system for design applied to the construction context. In *Proceedings of the CIB W78 workshop The life–cycle of Construction IT*.

Ekholm, A. and S. Fridqvist (1999, 7-8 June). The BAS•CAAD information system for design — principles, implementation, and a design scenario. In G. Augenbroe and C. Eastman (Eds.), *Computers in Building. Proceedings of the Eighth International Conference on Computer Aided Architectural Design Futures, CAADfutures'99*, Atlanta, pp. 149–164. Kluwer Academic Publishers.

Frege, G. (1994). *Funktion, Begriff, Bedeutung*. Vandenhoeck & Ruprecht in Göttingen.

Fridqvist, S. (2000). *Property–Oriented Information Systems for Design*. Ph. D. thesis, Division of Architectural and Building Design Methods, Lund University.

Galle, P. (1999, August). Artefact specification, design, and production as a process of communication. In *Proceedings of the InterSymp-99*.

Ganter, B. and R. Wille (1999). *Formal Concept Analysis*. Springer.

Garrett, Jr., J. H. and M. M. Hakim (1994, March). Class-centered versus object-centered approaches for modeling engineering design information. In *Proceedings of the IKM-Colloquium on Mathematics and Information Sciences in Building Engineering*, pp. 267–272.

Guarino, N. and C. A. Welty (2000). Ontological analysis of taxonomic relationships. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pp. 210–224.

Guénoche, A. and I. van Mechelen (1993). Galois approach to the induction of concepts. In *Categories and Concepts: Theoretical views and inductive Data Analysis*.

Haav, H.-M. (1997). An object classifier based on galois approach. In H. Kangassalo and others. (Eds.), *Proceedings of Information Modelling and Knowledge Bases, VIII*.

Hakim, M. M. and J. H. Garrett, Jr. (1994, June). Modeling engineering design information: An object-centered approach. In *Proceedings of the First ASCE Congress on Computing in Civil Engineering*, pp. 563–571.

RAISE Language Group, t. (1992). *The RAISE Specification Language*. Prentice Hall, CRI A/S.

RAISE Method Group, t. (1995). *The RAISE Development Method*. Prentice Hall, CRI A/S.

Shoemaker, S. (1997). Causality and properties. In D.H.Mellor and A. Oliver (Eds.), *Properties*, Oxford Readings in Philosophy. Oxford University Press.

Simon, H. A. (1981). *The Sciences of the Artificial*. MIT Press, Cambridge.

Simons, P. (1987). *Parts: A Study in Ontology*. Clarendon Press — Oxford.

Turk, Z. (1998). On theoretical backgrounds of cad. In I.Smith (Ed.), *Structural Engineering Applications of Artificial Intelligence, Lecture Notes in Artificial Intelligence 1454*, pp. 490–496. Berlin: Springer.

van Leeuwen, J. (1999). *Modelling Architectural Design Information by Features, and approach to dynamic product modelling for application in architectural design*. Ph. D. thesis, Eindhoven University of Technology, The Netherlands.