RULE-DEPENDENCIES FOR VISUAL KNOWLEDGE SPECIFICATION IN CONCEPTUAL DESIGN

Bodo Kraft[1] and Daniel Retkowitz[2]

ABSTRACT

Currently, the conceptual design phase is not adequately supported by any CAD tool. Neither the support while elaborating conceptual sketches, nor the automatic proof of correctness with respect to effective restrictions is currently provided by any commercial tool.

To enable domain experts to store the common as well as their personal domain knowledge, we develop a visual language for knowledge formalization. In this paper, a major extension to the already existing concepts is introduced. The possibility to define rule dependencies extends the expressiveness of the knowledge definition language and contributes to the usability of our approach.

KEY WORDS

Knowledge formalization, Conceptual Design, Graph Transformation

INTRODUCTION

A conceptual sketch describes in a coarse grained way the organization and functionality of a future building. Architects use this form of representation to get a first idea about the building and to better understand the requirements of its later usage, construction, and management. In contrast to conceptual design in civil engineering, the architectural conceptual design mainly regards the functional entities of the building, i.e. the rooms and areas, their function within the building and the relationships between them. The high level of abstraction enables the architect to rapidly evaluate different building sketches, to analyze the given alternatives, and to finally choose the best fitting sketch.

There are many restrictions that are already effective in the early design phase. Different domains, e.g. economy, law, functionality as well as the parties involved, like the building owner, the architect, the user, or the building contractor, prescribe restrictions and requirements to the future building. The conceptual relevant parts of the restrictions prescribe e.g. room and area sizes, room equipment, access and reachability relations.

In the ConDes project (Conceptual Design) at Aachen University of Technology we elaborate new concepts for tool support within the early phase of architectural design. We therefore develop a conceptual CAD tool as an extension of the industrial CAD tool ArchiCAD (GRAPHISOFT 2006, Kraft and Schneider 2005). Furthermore, we research how

[1] Dept. of Computer Science III, Aachen, University of Technology, Aachen, Germany, Phone +49/241/80-21314, FAX +49/241/80-22218, kraft@cs.rwth-aachen.de

[2] Dept. of Computer Science 3, Aachen, University of Technology, Aachen, Germany, Phone +49/241/80-21310, FAX +49/241/80-22218, retkowitz@cs.rwth-aachen.de

conceptually relevant knowledge can be easily formalized in a visual representation. In our idea, the formalization has to be done by a domain expert, i.e. by an architect or a civil engineer. Finally, we provide a way to use the defined knowledge to check a conceptual sketch by discovering restriction violations.

This paper is structured as follows. To give the reader an idea of the ConDes project, we firstly introduce our knowledge formalization approch, consisting of a domain ontology and predefined design rule types. All concepts are illustrated using hospitals as example for one class of buildings. Based on the fundamentals of our language for knowledge formalization, we introduce the new concept of rule dependencies in the main part of the paper. The provided tool support gives an idea about the implementation. Finally, we discuss related work and finish with a conclusion.

CONCEPTUAL DESIGN SUPPORT

In our approach, the support of conceptual architectural design is based on formalized knowledge which is represented by a set of design rules. These design rules impose certain restrictions on the future building and thus define a special class of buildings, namely the class of buildings that conform to the specified rules. The rules regarded in our case are of conceptual nature, i.e. they cover aspects like the internal organization of a building, e.g. the equipment and arrangement of rooms, the aggregation of rooms to cohesive areas, or the allowed respectively restricted traffic flow inside a building. Constructive details of the future building remain unconsidered in the conceptual phase as they are worked out in the later phases of the design process. The architect is supported by our tools in checking conceptual sketches of buildings against the specified design knowledge. This way the architect can find out whether there are any rule violations with respect to the requirements for the corresponding building class. In this paper we focus on the knowledge formalization part, especially on dependencies between individual design rules contained in a knowledge specification.

KNOWLEDGE FORMALIZATION

The formalized knowledge is not fixed inside our tool but instead the formalization is done dynamically at tool runtime by a domain expert, i.e. an architect or civil engineer with expertise in a special domain. In a static approach, all the relevant domain knowledge would have to be specified by the tool developers, which is neither feasible nor desirable. There are uncountable aspects that might be relevant in the design of a special class of buildings. Therefore it is not possible to take all these aspects into account at tool development time. Instead the specific domain knowledge has to be specified by domain experts as it is needed. In our approach the knowledge specification comprises two parts. The first part is the *domain ontology*. The domain ontology is used by the knowledge engineer to define the basic concepts that are needed to specify requirements of a class of buildings. The second part of the knowledge specification is a set of *design rules*. As already stated above, the design rules constitute the conceptual relevant knowledge that defines the restrictions that have to be obeyed in building designs of the respective class of buildings.

The structure of the domain ontology is predetermined by three basic elements, which can be further refined by the knowledge engineer:

- **Semantic objects** represent the conceptually relevant functional entities for the knowledge definition and the conceptual design,
- **Attributes** define properties of semantic objects in a conceptual design,
- **Relations** describe connections between semantic objects in a conceptual design.

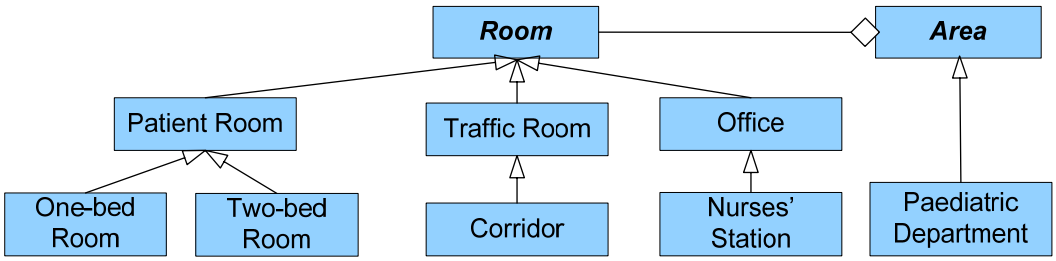

Figure 1: Definition of semantic objects in an example ontology

In Figure 1 an example design ontology is depicted. The knowledge engineer is supported in the definition of an ontology by several predefined semantic objects that are specific to the architectural domain, but still general and not related to a special class of building. These predefined concepts are *building site*, *building*, *storey*, *area*, *room*, and *section*. In the example of Figure 1 we can see the predefined concepts *room* and *area*. These are specialized by semantic objects that are specific to a certain class of buildings, in this case the class of hospitals.

After defining the domain ontology, the knowledge engineer proceeds with the formalization of design rules for the respective class of buildings. For this purpose, different types of design rules are offered by our tool. The most fundamental ones are *attribute rules*, *relations rules*, and *cardinality rules*:

- **Attribute rules** demand or forbid a certain property of semantic objects in a conceptual design, e.g. a size restriction or the demand respectively the forbiddance of certain equipment.
- **Relation rules** demand or forbid a certain connection between semantic objects, e.g. the access between two rooms.
- **Cardinality rules** restrict the number of occurrences of semantic objects in the building design.

There are several further rule types like aggregation rules, complex path rules, etc. Additionally we support advanced concepts like multiplicity restrictions, complex and runtime dynamic expressions, etc. These advanced rule types and concepts are beyond the scope of this paper. For further details we refer to (Kraft and Wilhelms 2004; Kraft and Wilhelms 2005).

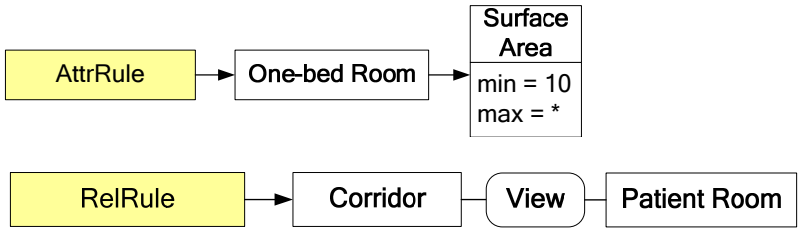

Figure 2: Example attribute and relation rule

In Figure 2 an example for an attribute rule and a relation rule is depicted. The upper part of the figure represents an attribute rule that demands a surface area for one-bed rooms of at least 10 sqm. The lower part represents a relation rule that demands a view connection between corridors and patient rooms.

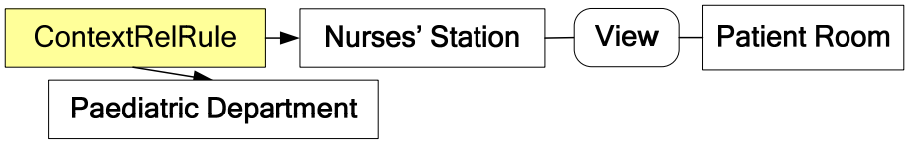

Figure 3: Example context relation rule

Figure 3 shows an example for a context relation rule. This rule type is an ordinary relation rule, but it is only evaluated when the concerned semantic object of the relation rule is aggregated within a semantic object of the specified type. In the above example, a view connection from nurses' stations in a paediatric department to a patient room is demanded.

CONSISTENCY ANALYSES OF CONCEPTUAL SKETCHES

To support the user in specifying an ontology, formalizing design rules, and creating a conceptual design we developed graph-based tools using the graph-rewriting system PROGRES (Schürr 1991) and the UPGRADE framework (Böhlen et. al. 2002). The operations to build up ontologies, design rules, and conceptual designs are implemented using graph-transformations that produce the respective nodes and edges in the run-time graph. The UPGRADE framework is used to provide an adequate visual representation and a user-friendly interface.

The tools used by a knowledge engineer to formalize conceptual knowledge are called *Domain Ontology Editor* and *Domain Knowledge Editor*. These tools allow the user to specify the required ontology concepts for a certain domain and to define respective design rules. The architect uses the *Design Editor* to create a conceptual sketch. The design editor is a graph-based CAD tool that allows for creating instances of semantic objects from the ontology, to assign properties to these instances and to define relations between them. Both parts, the conceptual design part and the domain knowledge part, can be checked automatically by the tools. This functionality is implemented using PROGRES graph

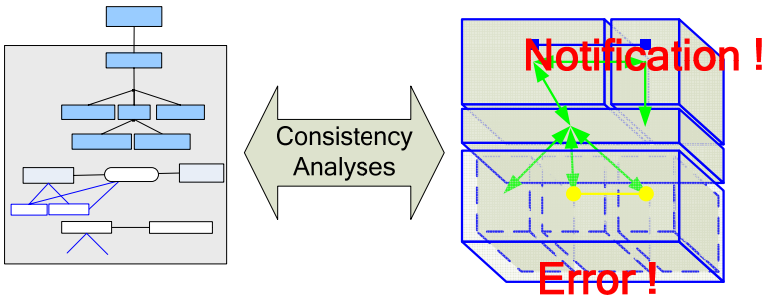

Figure 4: Consistency Analyses between a conceptual design and domain knowledge

transformations. That way, an architect can perform design checks on conceptual sketches and validate a sketch with regard to its conformity to the specified domain knowledge.

Figure 4 shows an example scenario for consistency analyses between conceptual knowledge and a conceptual sketch. On the left side of the figure, an ontology and some rules are illustrated. A rough sketch of a conceptual design is depicted on the right side of the figure. One can see several areas, represented by blue cuboids. Some of them – in this case the area-objects – contain further functional entities like rooms. The interrelationships between these elements are represented by differently colored arrows. After performing the design checks, the user is informed about possible rule violations in the conceptual design. Inconsistencies between a conceptual sketch and the formalized knowledge are visualized by notifications and error messages. Since we do not intend to narrow the creativity of architects when designing new buildings, we do not force the architect to solve all the inconsistencies. How to react on notifications is up to the architect, who can still decide whether a notification can be neglected or has to be solved at all costs.

RULE DEPENCIES

Although the expressiveness of the visual language already reached a comprehensive range, a large part of conceptually relevant knowledge could not be considered, because a lot of existing design rules did not only concern one semantic object in an atomic way. Many design rules are only valid under certain circumstances, others are grouped, so that all of them or just one have to be valid at the same time.

To bridge this gap, we introduced the concept of rule dependencies interrelating design rules. The basic concepts of the visual language, i.e. attribute rules, relation rules and cardinality rules, as well as the extended concepts like inheritance, aggregation and runtime-dynamic expressions are combined and interrelated with operators of Boolean and propositional logic (van Dalen 1997).

In the following, we will describe – considering some examples – the new elements of our visual language for knowledge formalization. We therefore start introducing the use of the unary operators *all* and *exist* and continue with the binary operators *and*, *or* and *implies*. The formal semantics does not differ from the common semantics of the logical operators. However their appliance in the field of knowledge formalization for conceptual design

strongly improves the expressiveness of the visual language, especially in combination with the existing design rules. The usability of the knowledge formalization approach by a knowledge engineer is still presevered. The tool support is presented in the next section.

The formal concepts of rule dependencies in visual knowledge formalization are based on propositional logic. The common unary operators constitute the first extension. Until now, each rule was, apart from the context restriction (ref. Figure 3), valid for all concerned semantic objects within a conceptual building sketch. Related to the propositional logic, the design rules were implicitly *all*-quantified. Looking at the effective conceptual knowledge, often restrictions or requirements are demanded that can already be fulfilled by one single building element. Therefore, we introduced new concepts to allow an explicit quantification of design rules. An *all-rule* allows defining that design rules have to be valid for all occurrences of the corresponding semantic object. In case of atomic and not further interrelated design rules, the semantics of the *all-rule* corresponds to that one of the existing design rules. As we see later, in more complex composed design rules, the *all-rule* allows an explicit linking to one certain semantic object. The semantics of the *exist-rule* differs. Here, the design rule is already fulfilled if at least one of the concerned semantic objects in a conceptual design meets the demands.

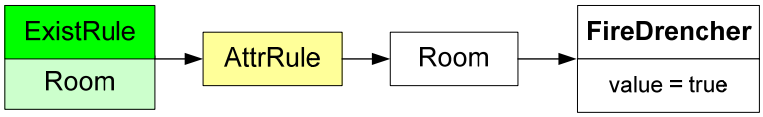

Figure 5: Boolean Attribute Rule with exist qantification

In Figure 5, a Boolean attribute rule demanding a fire drencher to be installed for any room is depicted, further qualified by an exist-rule. In the object-oriented domain ontology (ref. Figure 1) the semantic object *Room* is the root class. Thus, the attribute rule is implicitly valid for all instances of all subclasses of the semantic object *Room*. In a conceptual sketch, all rooms are demanded to have a fire drencher installed. The *exist-rule* changes the semantics of the design rules so that it is already fulfilled if at least one room in the conceptual sketch has the fire drencher installed. Again, because of the object-oriented domain ontology, the design rule is fulfilled independently from the room's type.

To further extend the expressiveness, we introduced the possibility to relate two design rules with a subset of the binary Boolean operators. Until now, all defined design rules had to be valid at the same time, i.e. there was no possibility to define alternatives or preconditions. Using the *and*-operator, one can now explicitly define a group of design rules, which have to be all valid at the same time. In contrast, the *or*-operator allows defining a set of design rules from which at least one has to be fulfilled. Finally, the *implies*-operator offers the possibility to express a precondition which has to be fulfilled before a design rules becomes effective, i.e. the design rule is only evaluated if the precondition holds.

In Figure 6 an example of a complex design rule, composed of two simple relation rules interrelated by an *implies*-operator, is depicted. The first relation rule expresses the forbiddance of an access between the toilet and the patient room. This one serves as precondition (assumption), i.e. only if this rule is fulfilled, the second design rule

(conclusion) is effective. The second relation rule demands access between the toilet and a vestibule. All together, the complex design in rule Figure 6 expresses that if the toilet has no access to the patient room, it has to have an access to a vestibule. The design rule is part of the German law, regulating the construction of hospitals (KhBauVO §29).

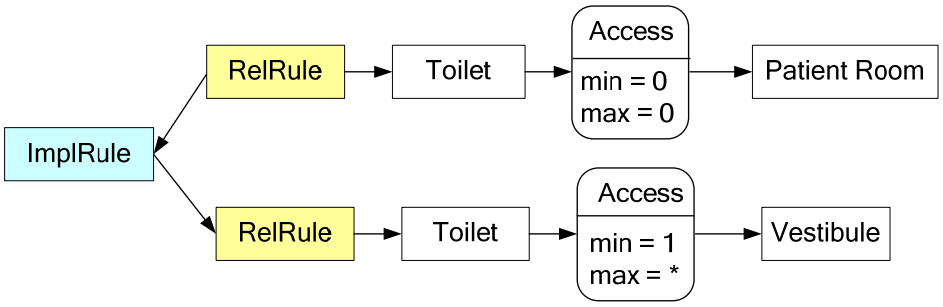

Figure 6: Implies rule with precondition

Of course, the new concepts of unary and binary operators to interrelate design rules can also be combined and nested. The formal semantics of the Boolean operators are the same as usual. Therefore the consistency analysis checking the complex design rules inductively interprets each design rule and afterwards its context. All existing functionality to check the atomic design rules is used unmodified. The evaluation of rule dependencies only operates on their results, i.e. if they are fulfilled or not. Therefore, arbitrarily nested complex design rules are (theoretically) possible.

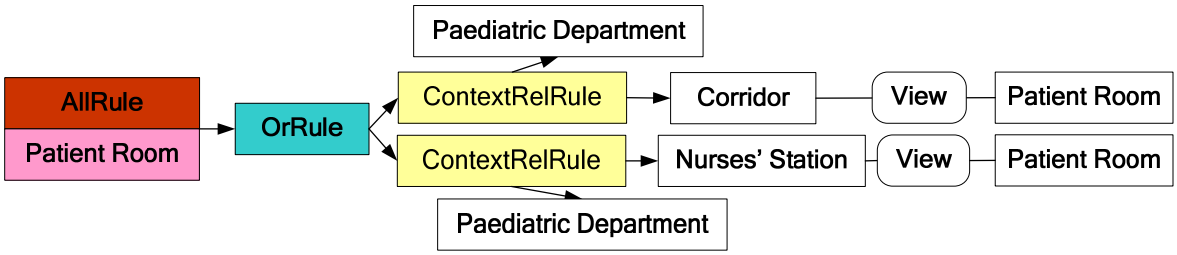

Figure 7: Complex design rule with *all*-quantification and *and*-operator

An example of a combination of design rules by operators and quantifiers is depicted in Figure 7. The complex rule is built-up from two context relation rules (ref. Figure 3), that demand a possibility to provide the view from the corridor and the nurses' station into the patient room. Because it is sufficient that one of the design rules is fulfilled, they are interrelated by an *or*-rule. Furthermore, for all patient rooms inside the conceptual building sketch one of these view relations has to be existent. This fact is expressed by the *all*-rule.

TOOL SUPPORT

The newly introduced concepts of dependencies between the existing elements of the visual language for knowledge formalization are fully implemented. Furthermore, the semantics of the consistency analyses checking a graph-based conceptual sketch against the defined knowledge is formally defined (Kraft and Retkowitz 2005) and implemented (Kraft and Retkowitz 2006). We use the graph-rewriting system PROGRES to specify a static graph schema and graph transformations to describe the dynamic aspects of a purely graph-based programm. The UPGRADE framework allows for generating visual graph-based tools which can be extended to the needs of a certain application domain. The tools PROGRES and UPGRADE are developed within a different project at our department. The complete tool construction process is described in (Böhlen et. al. 2002).

The *Domain Knowledge Editor* is the tool used by the knowledge engineer to input his domain knowledge. The provided tool support, using the basic elements of the visual language, has already been demonstrated and described in further papers (Kraft and Wilhelms 2005). These atomic design rules were defined within the so called *Object-Based View*. For adequately inserting and arranging simple and nested rule dependencies, a new view window called *Rule-Based View* was developed. The knowledge engineer uses this editor window to relate design rules with the introduced logical operators. A layout algorithm, optimized for tree structures, arranges the set of complex design rules in a consise matter. Additionally, a table summarizes all top-level design rules in a textual form, together with all associated documentation.

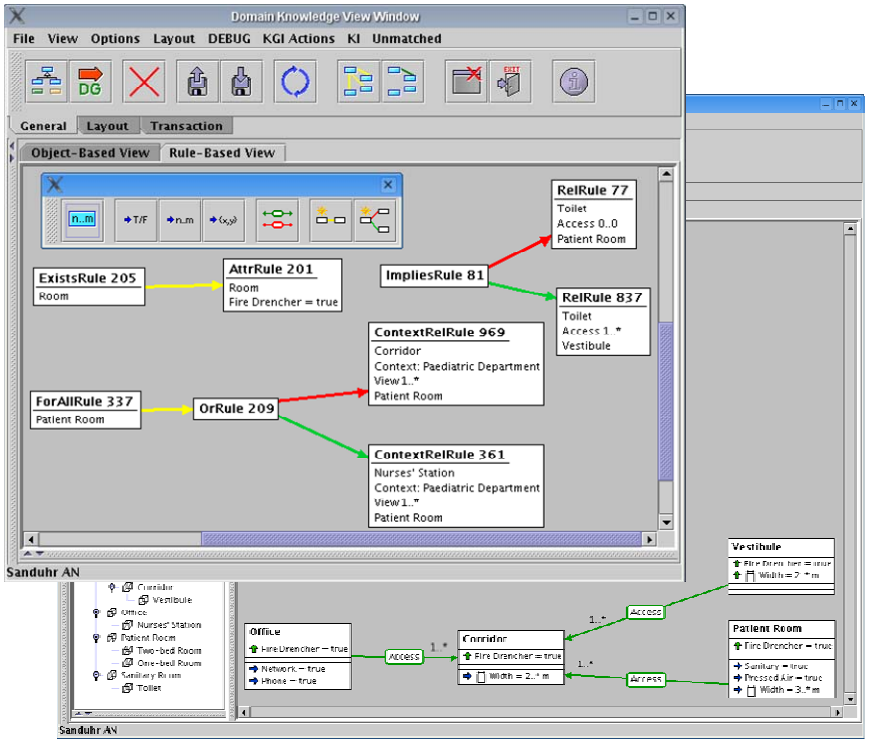

Figure 8: Screenshot displaying rule dependencies and atomic design rules

In Figure 8 two overlapping screenshots of our graph-based visual tools for knowledge formalization are depicted. In the background, the Object-Based View is depicted which allows to create and display atomic design rules. The term object-based indicates that all rules are grouped with respect to their concerned semantic object. Furthermore, all inherited design rules are visualized in a concise matter. None of the design rules depicted in this screenshot is part of a complex design rule, i.e. they are always effective. The design rules for the patient room e.g. demand, as previously introduced (ref. Figure 2), a sanitary installation, a minimum width and access to the corridor.

The second screenshot in Figure 8 depicts the Rule-Based View of the Domain Knowledge Editor. This new view-window allows for interrelating atomic and already interrelated design rules. The three previously introduced design rules are depicted in the screenshots, each graph node represents an atomic design rule or an operator, respectively.

To define and modify the domain ontology (ref. Figure 1), another visual tool, the *Domain Ontology Editor* exists. Furthermore, the visual tool *Design Graph Editor* allows for developing conceptual sketches and checking them against the defined knowledge.

RELATED WORK

There are several projects that concern the support of the early phase of the architectural design process. In (Gero 1990) the different semantics of the term *design* are discussed. It describes the product of the architects work on the one hand, and the process of developing the product on the other hand. We use the term conceptual design to describe the design process, and call the corresponding result a conceptual sketch. Christopher Alexander describes a way to define architectural design patterns (Alexander et al. 1977). Although design patterns are extensively used in computer sciences, in architectural design this approach has never been formalized, implemented and used. The SEED system (Flemming 1994) provides support for the early phase in architectural building design. In contrast to our approach, the SEED system focuses on the generation of sketches and not on an interactive design support. The importance of knowledge processing for architectural design is comprehensively discussed in (Coyne et al. 1990). In (Schmitt 1993), different new paradigms for a conceptual design support are proposed. Among other things, the top-down decomposition and modularization of sketches and the use of object-orientation for architectural design are introduced. Even if the work is neither implemented nor integrated into a CAD tool, the ideas are fundamental for our research.

In (Mora et al 2004) a framework is presented that supports civil engineers while finding the conceptual design of building structures. Different integration layers between the building structure and the conceptual design are discussed and an interactive approach is motivated. In contrast to this approach, we focus on the conceptual design in architectural design. In (Meniru et al. 2002) a tool is presented that works like a CAD system but additionally can identify the functional entities in a CAD sketch. Furthermore it is considered to check a sketch against knowledge. However, the tool is not implemented yet and no idea for the formalization and use of architectural knowledge is presented. Extracting all relevant information, concerning legal restrictions, from a 3D CAD model is the aim in (Sulaiman et al. 2002). This information should be used to check the model. However, none of that is implemented and it is not explained how the knowledge should be structured.

In (Steinmann 1997), an object-oriented approach for modeling architectural domains is presented. These models are intended to organize the whole design process. Therefore, Steinmann describes a generic process model for conceptual design. Like in our approach, Steinmann proposes a multilayered model concept for implementing model-based CAD tools. The generic process model is subdivided into several different phases. To each of these phases an atomic design action is assigned. The idea is to define the functionality of phase-specific tools this way. The atomic design actions are classified into control actions, synthesis actions, analysis actions, evaluation actions, and communication actions. Each of these action classes constitutes one phase, for which specific tools are to be built. We follow a different approach in our project. Our tools integrate the functionalities for conceptual design and are not split up into multiple phase-specific tools. We do not subdivide the conceptual design process into predefined sub-processes, because this would impose too strict limitations on the creative work of an architect in the early design phase.

In (Szuba 2005), another process model for conceptual design is proposed. At first, functional requirements and the structure of design elements are identified. For this reason, use cases and functional graphs are defined. The goal is to derive a prototypical design automatically from these use cases and functional graphs. In contrast to that, our tools do not generate conceptual designs automatically, because in our intention the architect's creative and artistic work should be supported by intelligent tools rather than be replaced, which in our view is not even possible in realistic complex scenarios.

CONCLUSIONS

In this paper, we described a knowledge-based support for the early phase of architectural design, called conceptual design. The main part of the paper is concerned with a major extension of our visual language for knowledge formalization. By providing the possibility to express interrelationships between design rules, the expressiveness of our knowledge formalization approach has been augmented.

The rule based definition and processing of design rules has contributed to the usability of our knowledge formalization approach. A survey of the German texts of law, design manuals and further sources has shown that a reasonable part of the conceptually relevant knowledge can be formalized using our visual language. Thus the expressiveness of our visual language for knowledge formalization matches the requirement.

With respect to the rule dependencies, it turned out that too deeply nested design rules sometimes become difficult for humans to understand. Even when the semantics of these constructs are clearly defined, they can be hard to retrace by humans. In practice however, most of the currently effective conceptual knowledge is built-up less complicated. Using two or three encapsulations of design rules, even the more complex regulations can be formalized. It also turned out that the visual definition of design rules is easy to understand and clearly readable if the knowledge base is not too big. Instead, if there are too many design rules displayed at the same time, the navigation through the defined knowledge becomes difficult.

Currently we are developing a support for different knowledge modules. Using them, base knowledge which has been formalized once can be used by different knowledge

engineers to define more specialized knowledge. These knowledge modules can be developed separately, reused and integrated to a complex knowledge base. We expect that sectioning different knowledge modules will also solve the problem of navigating through large knowledge bases and improve the readability of the formalized knowledge.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of this project by the German Research Foundation (DFG) within the scope of the priority program ”Network-based Co-operative Planning Processes In Structural Engineering” (Meissner et al. 2006).

REFERENCES

Alexander, C., Ishikawa, S., and Silverstein, M. (1977). *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press.

Böhlen, B., Jäger, D., Schleicher, A. and Westfechtel, B. (2002). “UPGRADE: A Framework for Building Graph-Based Interactive Tools”. *Proc. 1st Intl. Conf. on Graph Transformation* (ICGT’02), LNCS 2505, pp. 270–285. Springer.

Coyne, R. D., Rosenman, M. A., Radford, A. D., Balachandran, M., and Gero, J. S. (1990). *Knowledge-Based Design Systems*. Addison Wesley.

Flemming, U. (1994). “Case-Based Design in the SEED System”. *Knowledge-Based Computer-Aided Architectural Design*. pp. 69–91. Elsevier.

Gero, J. S. (1990). “Design Prototypes: A Knowledge Representation Schema for Design”. AI Magazin, 11(4), pp. 26–36. AAAI Publisher.

GRAPHISOFT. (2006). *GRAPHISOFT Homepage*. www.graphisoft.com.

Kraft, B. and Retkowitz, D. (2005). „Operationale Semantikdefinition für konzeptuelles Regelwissen“. *Proc. Forum Bauinformatik 2005*, pp. 173–182. Department Bauinformatik BTU Cottbus, Germany.

Kraft, B. and Retkowitz, D. (2006). “Graph Transformations for Dynamic Knowledge Processing”. *Proc. of the 2006 Intl. Conf. on System Sciences (HICSS 2006)*, pp. 1–10. IEEE Press.

Kraft, B. and Schneider, G. (2005). “Semantic Roomobjects for Conceptual Design Support”. *Proc. of the 11th Intl. Conf. on Computer Aided Architectural Design Futures (CAAD Futures 2005)*, pp. 207–216. Springer.

Kraft, B. and Wilhelms, N. (2004). “Interactive Distributed Knowledge Support for Conceptual Building Design”. *Proc. of the 10th Intl. Conf. on Computing in Civil and Building Engineering (ICCCBE-X)*, pp. 1-14. ASCE.

Kraft, B. and Wilhelms, N. (2005). „Visual Knowledge Specification for Conceptual Design“. *Proc. of the 2005 Intl. Conf. on Computing in Civil Engineering (ICCC 2005)*, pp. 1–14. ASCE.

Meissner et al. (2006). “DFG - Priority Program 1103 Network-based Co-operative Planning Processes”. www.dfg-spp1103.de.

Meniru, K., Bedard, C. and Rivard, H. (2002). “Early Building Design using Computers”. *Proc. of the Conf. on Distributing Knowledge in Building (CIB w78 2002)*. Aarhaus School of Architecture.

Mora, R., Bedard, C., and Rivard, H. (2004) "A Framework for Computer-Aided Conceptual Design of Building Structures." *Proc. of the 10th Intl. Conf. on Computing in Civil and Building Engineering (ICCCBE-X).* Bauhaus-University Weimar.

Schmitt, G. (1993). *Architectura et Machina – Computer Aided Architecural Design und Virtuelle Architektur*. Vieweg.

Schürr, A. (1991). *Operationales Spezifizieren mit programmierten Graphersetzungs-systemen.* Dissertation. Aachen, University of Technology.

Steinmann, F. (1997). *Modellbildung und computergestütztes Modellieren in frühen Phasen des architektonischen Entwurfs*. Dissertation. Bauhaus-University Weimar.

Sulaiman, M. J., Weng, N. K., Theng, C. D. and Berdu, Z. (2002). "Intelligent CAD Checker For Building Plan Approval". *Proc. of the Conf. on Distributing Knowledge in Building (CIB w78 2002)*. Denmark. Aarhaus School of Architecture.

Szuba, J. (2005). *Graphs and Graph Transformations in Design in Engineering*. Dissertation. Darmstadt University of Technology.

van Dalen, D. (1997). *Logic and Structure*. Springer.