Proc. of the 2005 ASCE Intl. Conf. on Computing in Civil Engineering (ICCC 2005)
eds. L. Soibelman und F. Pena-Mora, Seite 1-14, ASCE (CD-ROM), Cancun, Mexico, 2005

VISUAL KNOWLEDGE SPECIFICATION FOR CONCEPTUAL DESIGN

Bodo Kraft[1], Nils Wilhelms[2]

ABSTRACT

Current CAD tools are not able to support the fundamental conceptual design phase, and none of them provides consistency analyses of sketches produced by architects. To give architects a greater support at the conceptual design phase, we develop a CAD tool for conceptual design and a knowledge specification tool allowing the definition of conceptually relevant knowledge. The knowledge is specific to one class of buildings and can be reused. Based on a dynamic knowledge model, different types of design rules formalize the knowledge in a graph-based realization. An expressive visual language provides a user-friendly, human readable representation. Finally, consistency analyses enable conceptual designs to be checked against this defined knowledge. In this paper we concentrate on the knowledge specification part of our project.

KEYWORDS

Visual Language, Knowledge Specification, Conceptual Design, Graph Transformation

INTRODUCTION

The difficulty and complexity of construction processes coupled with the wider and increasing distribution and interaction of the players involved in these processes means that currently existing CAD tools are no longer able to provide adequate support in their execution. Currently, there is no CAD tool available for supporting the fundamental conceptual design phase, and none of the currently available CAD tools enables consistency analyses of sketches produced by architects.

To provide architects with greater support at the conceptual design phase, there is a need for CAD tools to be extended from their currently detailed (yet limited) constructional design application. Rather than thorough and detailed plans, conceptual design focuses on the functionality of buildings as a whole, consisting of various functional, interrelated entities. This abstraction allows identification of the organizational configuration of a building and ensures its usability.

Even when the conceptual design phase is executed at the beginning of a building construction process and the degree of detail remains low, account needs to be taken of a large number of restrictions arising from various domains, such as legal aspects, technical restrictions, functional specifications, and financial restrictions. The majority of these restrictions are specific to one class of buildings, e. g. office buildings, car garages or residential build-

[1] Department of Computer Science III, Aachen University of Technology
D-52074 Aachen, PH +49/241/8021314
kraft@i3.informatik.rwth-aachen.de
[2] nilsw@i3.informatik.rwth-aachen.de

ings. Since the conceptual design takes place at the beginning of the building construction process, a consistent sketch forms the basis of all following design stages.

In the next step architects manually transfer the conceptual design, usually in the form of a hand drawing, into a CAD tool. Today existing CAD tools can only handle *constructive design information*, e. g. all kinds of used materials, constructive elements like walls and doors, and their exact dimensions. Thus, the valuable *conceptual design information*, i. e. all decisions about the organizational structure of the building, gets lost.

The complete scenario of our project comprises two approaches. First, we formally specify a CAD tool, the *Design Graph Editor*, for conceptual design via graph technology, based on the graph rewriting system PROGRES (Schürr 1991) and the UPGRADE framework (Böhlen et al. 2002). A further graph-based tool, the *Knowledge Graph Editor*, allows the definition of conceptually relevant knowledge specific to one class of buildings. *Consistency analyses* enable conceptual designs to be checked against this defined knowledge with notifications to the architect when restrictions are violated (Kraft and Wilhelms 2004; Kraft et al. 2002). These graph-based tools can be seen as research prototypes. Even when it provides a graphical user interface and is user-friendly, there is always an effort learning a new tool. For this reason, in the second part of our project (Kraft and Schneider 2005), by way of example we extend the CAD tool ArchiCAD (GRAPHISOFT 2005) with new functionality for conceptual design and consistency analyses.

In this paper we concentrate on the knowledge specification part of the project, introducing a model architecture for graph-based knowledge specification, conceptual design and parameterized consistency analyses (Kraft and Nagl 2004). Based on this architecture, in the main part of the paper we describe a new visual language for graph-based knowledge definition specific to the domain of conceptual building design. We outline both, the power of expression of this visual language and its graph-theoretical background.

SYSTEM ARCHITECTURE

The system architecture describes the organization of the complete project and the dependencies between its parts, to illustrate how the knowledge specification is embedded in the context of the whole project.

Figure 1 depicts the system architecture of the complete scenario for conceptual design support. The two main sections in the scenario, separated by vertical columns, are the *knowledge specification* part on the left hand side and the *conceptual design* part on the right hand side. Furthermore, the system architecture is structured into three horizontal layers. On top, the *implementation* with PROGRES is depicted, in the middle layer the *knowledge definition* part and below the developed *tool support*. The arrows depicted between the columns describe the necessary integration between knowledge and design in each layer.

The first layer at the top of Figure 1 stands for a graph grammar specification using the graph rewriting system PROGRES. In this layer, a *graph schema* determines graph node classes, node attributes, and edge types to restrict the valid graph class to the needed subset (Kraft and Nagl 2003; Kraft and Wilhelms 2004). Additionally, the PROGRES specification comprises *graph transformations* that determine valid modifications of a graph, they provide the realized functionality. The graph transformations can be executed, to build-up and modify a host graph, representing the underlying data structure for modeling a certain application

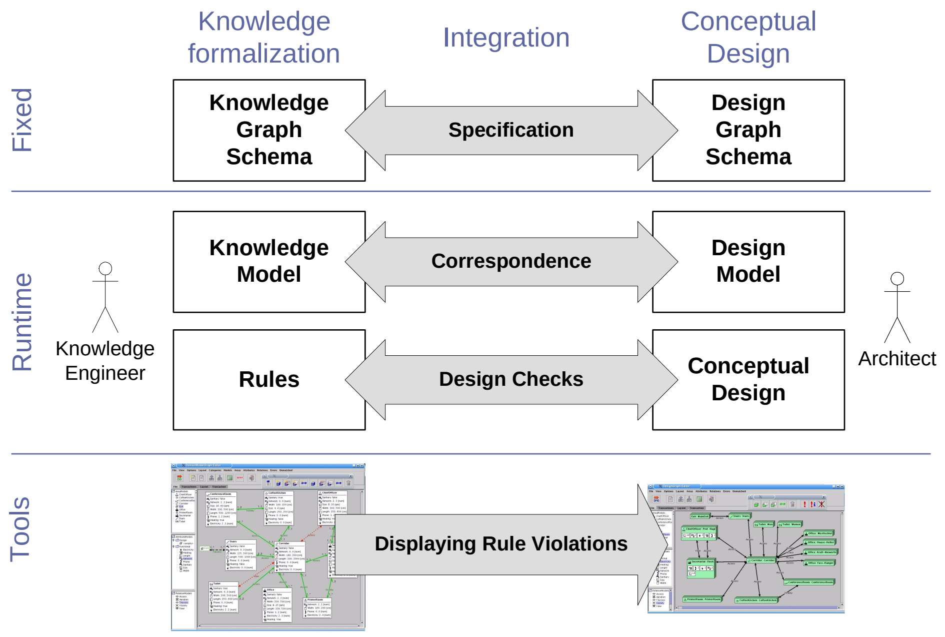

Figure 1: System Architecture and Tool Support

domain. In this layer, knowledge and design are combined inside the PROGRES specification as they are built-up from a similar internal graph representation. Integration between knowledge and design is therefore already given. The PROGRES specification, as well as the extensions to the UPGRADE framework, are *fixed by the tool developer*; they do not comprise any building class specific knowledge.

The second layer encloses the runtime dynamic part of the system architecture. On the left hand side, a domain expert, usually an experienced civil engineer or architect formalizes conceptual knowledge specific to a certain class of buildings. We call this domain expert *knowledge engineer*. The knowledge formalization part is again subdivided into two layers, namely the definition of the *knowledge model*, and based hereon the definition of *design rules*

The *knowledge model* serves as a definition and classification of the relevant concepts for knowledge formalization in conceptual design. The knowledge model is specific for one class of buildings. Among other things, different room and attribute types are defined here, they serve as atomic components for the following specification of design rules. In our system architecture, the definition of the knowledge model is done *at tool runtime* by a knowledge engineer and is not fixed in the implementation, because knowledge formalization has to be done by a domain expert (not the tool developer). Furthermore, the formalization is an evolutionary process with much iteration, so that a fixed knowledge model definition would not be applicable.

Based on the defined concepts, the actual domain knowledge in the form of *design rules* is defined by the knowledge engineer. We therefore provide a graph-based visual language for knowledge specification which allows defining attribute, cardinality and relation rules, complex relations and parameterized rules in form of runtime depending expressions (see section Design Rules). The effort of knowledge formalization only pays off, if several projects of the same class of buildings are conducted.

Looking at the right hand side of the second layer in Figure 1, the same architecture is depicted, now for the *conceptual design of buildings*. To support architects during conceptual design, we follow two approaches. First, we extend the industrial CAD tool ArchiCAD with new functionality. In this case, the design model is determined by the CAD tool developer, in our case GRAPHISOFT. A less proprietary example of a design model would be the IFC-interface, developed to improve the portability of architectural sketches. Second, we develop an own graph-based CAD tool for conceptual design support. Here, the design model can be elaborated at runtime, analogously to the knowledge model. The conceptual design is elaborated in the first case by an architect using ArchiCAD (Kraft und Schneider 2005); in the second case the architect creates a graph-based conceptual design.

In both cases, an *integration document* (Becker and Westfechtel 2004) between the *knowledge model* on the left hand side and the *design model* on the right hand side has to be defined, to determine the corresponding model elements. The integration document is essential for checking the consistency between the design rules and the conceptual design by graph-based analyses. The model-integration is not the main focus of our work; we currently just use a simple integration solution.

Finally, in the last layer the developed tools are depicted. The graph-based tools are derived from the PROGRES specification. The UPGRADE framework provides a reusable and extensible platform for executing such graph-based programs in a problem adequate representation. For knowledge specification we develop a graph-based visual application, used by the knowledge engineer, the *Knowledge Graph Editor (KGE).* The KGE provides functionality to elaborate a knowledge model and to define the actual domain knowledge in form of design rules. The knowledge engineer is supported by problem adequate views on the knowledge graph and by layout algorithms displaying the graph in an ordered matter. For conceptual design we develop the *Design Graph Editor (DGE)*. This graph-based CAD tool allows elaborating conceptual sketches corresponding to the defined design model. Using the DGE, the architect can concentrate on the relevant functional entities, their attributes and interrelationships. By executing the graph-based *Design Checks,* inconsistencies between the knowledge specification and the conceptual design are discovered and visualized inside the DGE to inform the architect.

A VISUAL LANGUAGE FOR KNOWLEDGE SPECIFICATION

In this section, we describe the new developed visual language for knowledge specification. Because the dynamic *knowledge model definition* is an essential part of the knowledge formalization process, we start by introducing a knowledge model as an example for explaining the expressiveness of the actual knowledge definition. A more complete description of the internal realization of the knowledge model can be found in (Kraft and Wilhelms 2004). Because our knowledge formalization is based on object-oriented concepts, we use UML like diagrams for representation.

KNOWLEDGE MODEL

The knowledge engineer uses the knowledge model to define and classify the relevant concepts for knowledge specification. It contains three groups of elements, which are defined in the graph schema: *semantic objects* describe functional entities of a building, e. g. rooms; *relations* describe relationships between them, e. g. access between two rooms; finally *attributes* describe properties of the functional entities, e. g. what size a room should have.

We distinguish three types of semantic objects: a *room* is defined as a usually wall enclosed element; a *section* is a part of a room, and an *area* is an aggregation of several rooms or again areas. The *inheritance* relationship between semantic objects is expressed by an arrow with a white head. In Figure 2 one can see a cut-out of a sample knowledge model for car garages. The middle part contains the definition of the semantic objects. Looking at Figure 2, the semantic object "Room" is the root class of all rooms. It has three subclasses, "Motor Vehicle Room", "Customer Room" and "Sanitary Room" and so on.

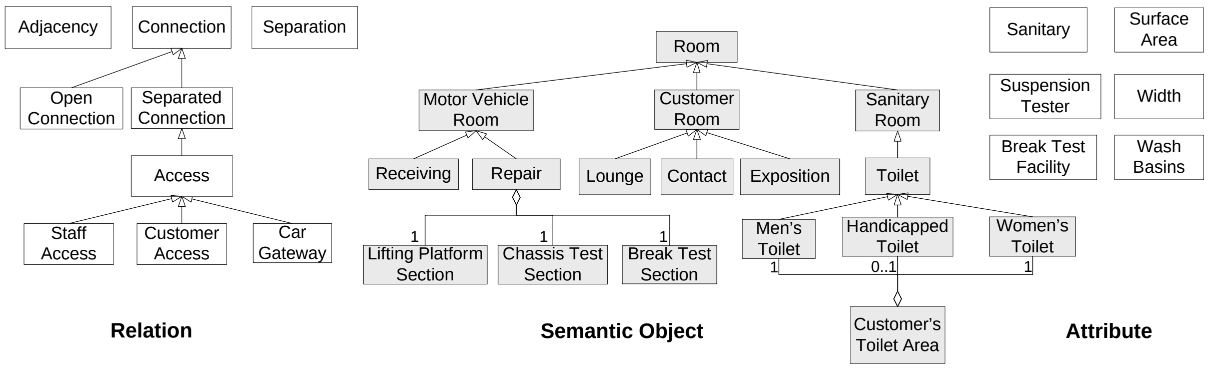

Figure 2: Knowledge Model, specific for Car Garages

The *aggregation* relationship between an area and several rooms is represented by a connection edge with a diamond at its beginning. At the end of an aggregation edge the number of contained semantic objects is shown in form of UML multiplicities. The "Customer Toilet Area", e. g., must contain one men's toilet ("1"), one women's toilet ("1"), and can optionally contain a handicapped toilet ("0..1").

To express relationships between semantic objects the knowledge model also contains relations (Figure 2, left hand side). In the example knowledge model one can see three root relations: "Adjacency", "Connection" and "Separation". While "Adjacency" and "Separation" express that two semantic objects should be neighbored and separated, respectively, the "Connection" relation is specialized into an "Open Connection", e. g. no wall in between, and a "Separated Connection", e. g. a door. Finally, the separated "Access" connection is specialized into three further types of connection determining in which way it is used.

On the right hand side of Figure 2 six attributes are defined. "Sanitary" is a Boolean attribute describing if a semantic object should have a sanitary installation ("true" valued) or not ("false" valued). The "Surface Area" determines the size of a semantic object; it's allowed values in the later design are defined by an integer range.

DESIGN RULES

The knowledge engineer defines design rules based on the knowledge model (see Figure 2). Design rules instantiate the defined components. Each design rule concerns one semantic object. There are three basic types of design rules:

- *Attribute rules* demand semantic objects to have particular properties (s. section Attribute und Relation Rules).
- *Relation rules* define the obligation or prohibition of the existence of certain relationships between semantic objects (s. section Attribute und Relation Rules).
- *Cardinality rules* restrict the number of actual semantic objects of one class in the conceptual design (s. section Cardinality Rules).

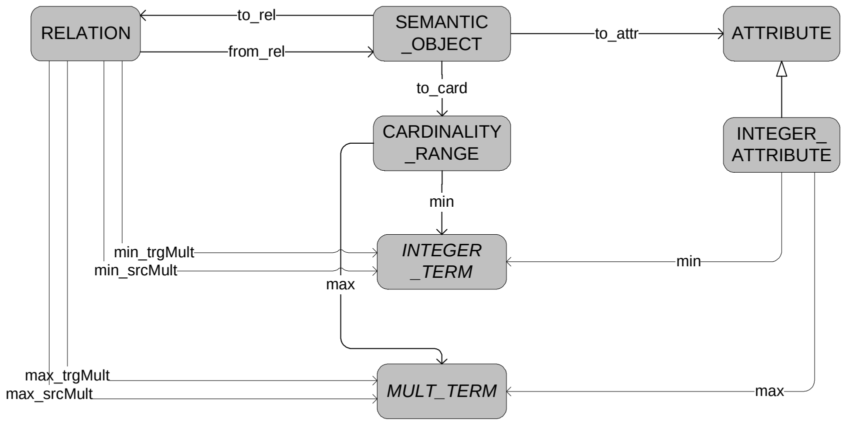

Figure 3: Cut-out of Graph-Schema for Design Rules

In the graph schema, the composition of design rules is formally specified. Each design rule is composed of one semantic object, and corresponding to the type of rule, a relation with a second semantic object, an attribute, or a cardinality range.

A simplified cutout of the graph schema is depicted in Figure 3. An attribute rule is composed of a “SEMANTIC_OBJECT” connected via a “to_attr”-edge with an “ATTRIBUTE”. The constituent parts of a relation rule are two “SEMANTIC_OBJECTS”, interrelated with two edges, “to_rel” and “from_rel”, with a “RELATION”. Finally, a cardinality rule consists of a “SEMANTIC_OBJECT” and a “CARDINALITY_RANGE”, with a “to_card”-edge in between.

For knowledge specification *integer expressions* are needed in several places: an “INTEGER_ATTRIBUTE” defines a range of values, valid for numericallydescribable attributes, e. g. length between 4 and 6 m. The realization of this range in the graph schema is defined by an “INTEGER_TERM” as lower bound, and a “MULT_TERM” for the upper bound. An integer term can be a constant integer literal or a complex integer expression. A “MULT_TERM” can either be an integer term or the star symbol which allows arbitrarily values. In the same way, minimal and maximal multiplicity restrictions of a relation and a valid cardinality range for semantic objects are formalized using integer expressions, based on “INTEGER_TERM” and “MULT_TERM”.

In the following, the characteristics of the visual language are introduced. For a better readability, the concepts are represented in a condensed form that gives an abstraction from the internal graph-based realization.

Attribute und Relation Rules

Attribute and relation rules constitute the basis of our knowledge specification approach. In conceptual design, a precise description of semantic objects and their relationships is essential.

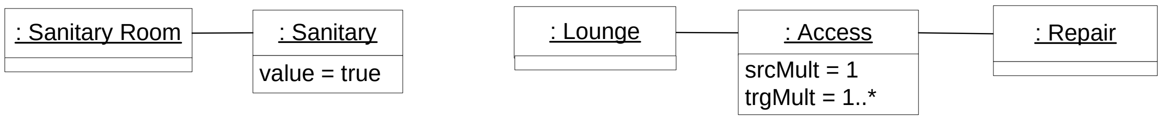

Figure 4: Attribute and Relation Rules

As mentioned before, attribute rules determine valid *properties* of semantic objects in the design. There are three types of attribute rules which differ in the used data type. *Integer attributes* describe an interval of valid values, e. g. size of a room. *Boolean attributes* determine the availability of certain equipment, e. g. sanitary installation. Finally, *enumeration attributes* define a set of valid string values, e. g. orientation of a room to “north”, “south”, “west” or “east”. On the left hand side of Figure 4 a sample Boolean attribute rule is shown. The rule demands each “Sanitary Room” to have a “Sanitary” installation, in order to guarantee fluent water.

Relation rules determine the existence or non existence of *relationships between semantic objects* in the conceptual design. The number of semantic objects connected by a relation is restricted. The declaration of the source and target multiplicities allows a precise definition of how two semantic objects can be interrelated. A target multiplicity greater than zero as lower bound demands each semantic object of the source class to be minimally interrelated with the specified number of semantic objects of the target class. The upper bound restricts the maximally allowed number of connected semantic objects. A special case forms a zero multiplicity for lower and upper bound: Such a rule prohibits the existence of a relation between the corresponding semantic objects. Analogously, the source multiplicity restricts the number of connected semantic objects of the target class.

The relation rule depicted on the right hand side of Figure 4 demands the existence of an “Access” relation between the “Lounge” and a “Repair” room, thus a customer can show the garage foreman the location of his car's problem. To determine that a “Repair” room has only access to exactly one “Lounge”, the source multiplicity (“srcMult”) has the value “1”. On the other hand the “Lounge” should have access to at least one “Repair” room, so the target multiplicity (“trgMult”) is “1..*”.

Cardinality Rules

In conceptual design, the number of occurrences of semantic objects is important for the building organization. Again, we distinguish if a semantic object is essential to guarantee the functionality of the building, if a semantic object can optionally occur, and if a semantic object must not exist in the future building. A regulation is possible by defining the lower and upper bound of a cardinality rule. The lower bound determines if a semantic object is optionally (= 0) or essential (> 0), while the upper bound defines the maximal number of allowed occurrences of semantic objects. The upper bound is represented by a constant integer value or an integer expression, including the star symbol.

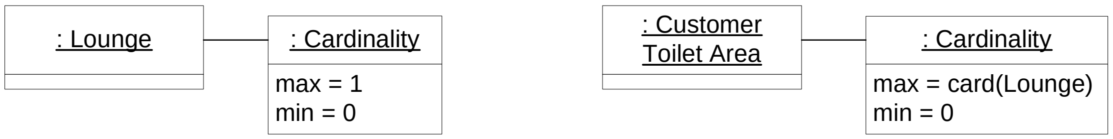

Figure 5: Cardinality Rules

Regarding the example of a car garage again, a customer lounge can be optional and the maximal number of lounges is one. This fact is expressed by the cardinality rule on the left hand side of Figure 5: since a lounge is optional the “min” value of the cardinality range is set to “0”; the “max” value is set to “1”. On the right hand side of Figure 5 another example of a cardinality rule is given. Again, a customer toilet area is optional as the minimal number is “0”. The maximal number, however, depends on the actual number of lounges, which can be “0” or “1”, as seen before. Thus, the semantics of this rule can be stated by: “There can be one customer toilet area, but only if there is a lounge”. This rule already uses the later introduced concept of *runtime dynamic expressions*.

Inheritance and Aggregation

As presented in Figure 2, the knowledge model follows an object oriented classification. The semantics of the described design rules are thereby extended by inheritance of semantic objects. Design rules, specified for a semantic object, are propagated to all inheriting semantic objects in the knowledge model hierarchy. This mechanism allows defining *common knowledge* on a general level and reduces the effort of knowledge specification. Analogously to the object oriented concepts, design rules can also be redefined if necessary.

The analyses and structuring of inheritance for knowledge specification is realized in the PROGRES specification by special nodes and edges. Complex graph transformations, operating on these graph elements, ensure the consistency of the object oriented structuring. The internal realization, however, is hidden from the user.

The example in Figure 6 demonstrates the semantics of inheritance. The attribute rule for “Sanitary Room” demands the surface area to be between 10.0 and 15.0 square meters (“Defined” in Figure 6). Regarding the knowledge model cut-out one can see that the “Sanitary Room” is specialized into a “Toilet” which itself is specialized into particular types of toilets. The above defined attribute rule is now stepwise inherited by all subclasses (“Derived”). The attribute rule for “Handicapped Toilet”, however, is redefined to guarantee a minimum turning circle for wheelchairs. All semantic objects inheriting from “Handicapped Toilet” must then fulfill the redefined rule.

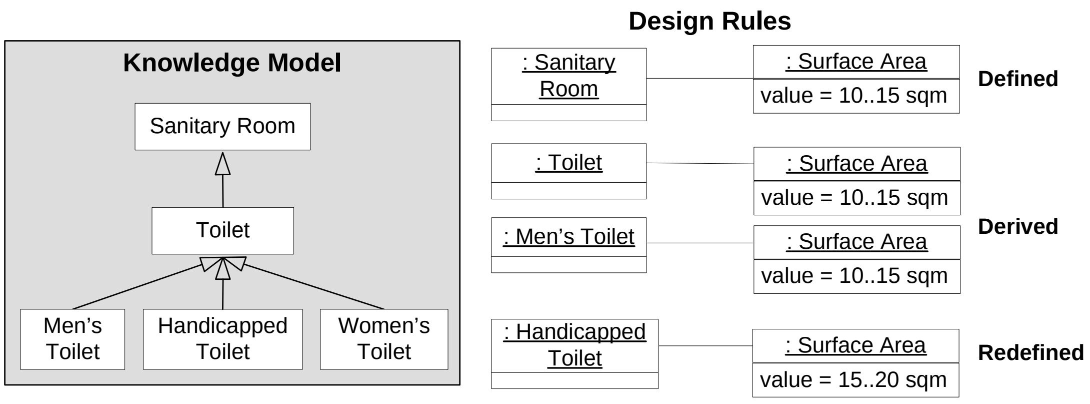

Figure 6: Inheritance and Redefinition of Design Rules

A further concept appointed in the knowledge model definition is the *aggregation* of semantic objects. Using the aggregation relation, *complex semantic objects* can be defined, composed of a collection of simple or again complex semantic objects. In contrast to the object oriented classification mechanism, which describes a homogeneous set of semantic objects with similar properties, aggregation usually combines a heterogeneous set of different semantic objects. In conceptual design several constraints concerning these aggregations are relevant. We therefore provide the functionality to define attribute, relation, and cardinality rules for complex semantic objects, too. To archive unambiguous semantics, further information is needed to determine how such a design rule is interpreted. We use a "mode" flag to distinguish different evaluation alternatives: "one", "all", and for integer attributes additionally "sum", "max" and "min". If the "mode" flag is set to "one" or "all", the design rule is effectual for one single semantic object inside the aggregation, or all of them, respectively. If the "mode" flag of an integer attribute is set to "sum", the design rule prescribes the summation of all concerned integer attribute values of the semantic objects inside the aggregation.

Regarding the example in Figure 7, a cut-out of the knowledge model and three example design rules with a complex semantic object are depicted. In the knowledge model, a "Customer Area" is composed of optionally one "Lounge", one "Contact" room, one "Exposition" room, and one optional "Customer Toilet Area". To restrict the surface area of this aggrega-

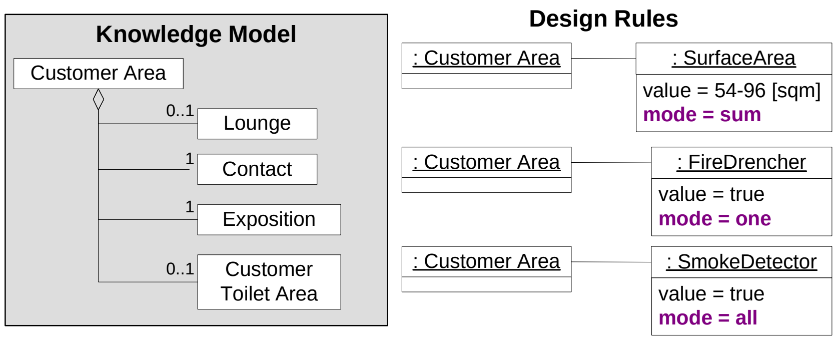

Figure 7: Design Rules with Aggregation

tion to be between 54 and 96 square meters, an attribute rule concerning the “Customer Area” is defined (Figure 7, first design rule). The “mode” flag is set to “sum”, therefore the summation of the surface areas of all aggregated semantic objects in the conceptual design must be within the defined interval. The second and third rule demand at least *one* semantic object in the aggregation to be equipped with a fire drencher, and *all* with a smoke detector, respectively.

Complex relation rules

As an extension to relation rules, which determines mandatory and forbidden intermediate relationships between two semantic objects, we further introduce complex relation rules. Especially in conceptual design, composed relations are needed to express advanced concepts like transitive accessibility or restricted reachability.

Complex relations are realized as a concatenation of semantic objects and relations. The semantic objects at the beginning and the end of a complex relation definition restrict the application of connecting semantic objects to the determined class and its subclasses in the knowledge model. The inner components of the complex relation definition determine its internal built-up.

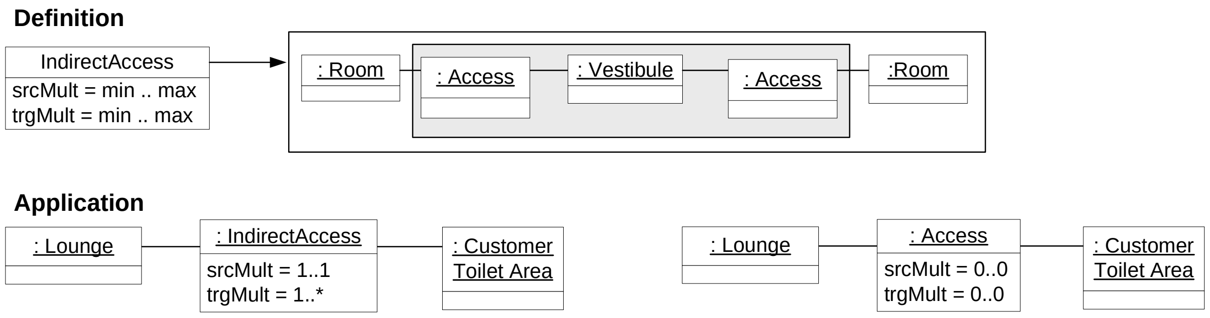

Figure 8: Complex Relation Rules: Definition and Application

An example of a complex relation design rule is depicted in Figure 8. The definition of the complex relation “IndirectAccess” demands that the access from one room to another contains an intermediate vestibule, e. g. to avoid noise pollution or bad smell. Below the definition in Figure 8 an application of this complex relation is given. To avoid a direct access between the lounge and the customer toilet area two design rules are defined. The first one, a complex relation rule, *demands* an *indirect* access between the “Lounge” and the “Customer Toilet Area”. Additionally, the second rule, a simple relation rule, *forbids* the *direct* access between the two semantic objects. Thus, the customer toilet area is only allowed to be accessible through an intermediate vestibule.

Runtime dynamic expressions

We previously introduced a cut-out of the graph schema as the internal realization of design rules (Figure 3) and briefly motivated the need for *runtime dynamic expressions*. With this concept we provide a powerful possibility to define complex integer and Boolean expressions. These dynamic expressions are evaluated at runtime in dependency to an actual build-

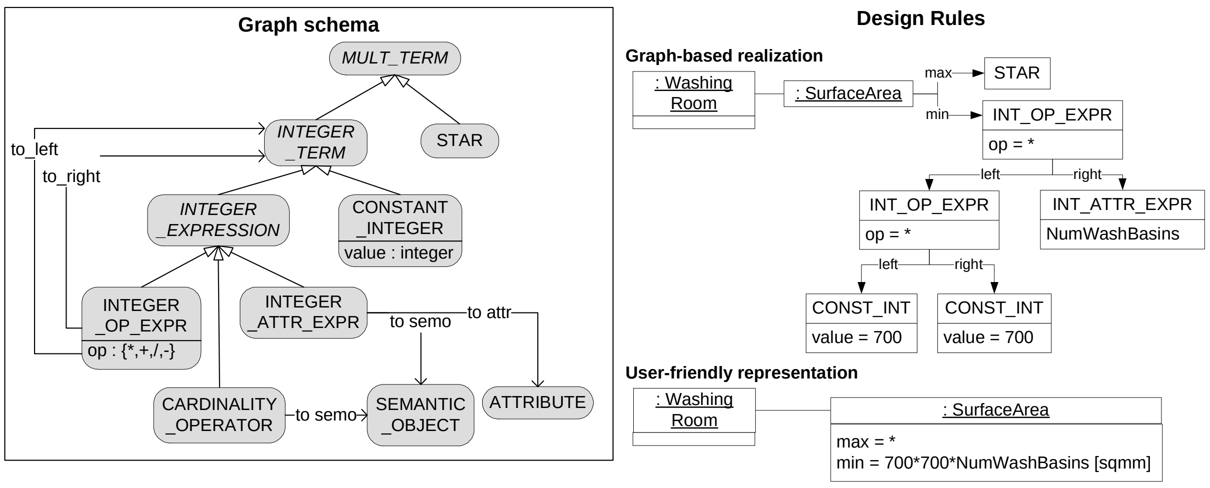

Figure 9: Runtime Dynamic Expressions, Graph Schema and Example Rule

ing design. Runtime dynamic expressions do not introduce a new kind of design rule, but extend the existing expressive power of the visual language. The concept of expressions is fundamental because it is applied to the definition of attribute values and relation multiplicities, as well as cardinality ranges.

On the left hand side of Figure 9 the graph schema for runtime dynamic integer expressions is depicted. The root class “MULT_TERM” is used to define the upper bound of an integer range, thus upper bounds can be an “INTEGER_TERM” or a “STAR” (infinite). Lower bounds can only be integer terms, as the star symbol is not allowed to be used. An “INTEGER_TERM” is specialized into a “CONSTANT_INTEGER” to represent a static integer literal, and a composite “INTEGER_EXPRESSION”. We distinguish three different specialized integer expressions:

- An integer operator expression (“INTEGER_OP_EXPR”) represents an arithmetic expression, e. g. the multiplication of two integers. It consists of an arithmetic operator (*, +, /, –), and a left and a right operand of type “INTEGER_TERM”. The inductive definition allows arbitrarily nested operator expressions.
- An integer attribute expression (“INTEGER_ATTR_EXPR”) refers to an actual attribute value of a semantic object in the conceptual design.
- A cardinality operator (“CARDINALITY_OPERATOR”) computes the actual number of semantic objects of one class in the conceptual design (see Figure 5 for an example). In the graph schema the operator references a semantic object.

An example application of a runtime dynamic expression is shown in Figure 9 on the right hand side. The German law for working places (ArbStättV §35 (3)) demands a minimum surface area for washing rooms depending on the number of wash basins in the room. For each wash basin there must be a minimal surface area of 0.7 times 0.7 square meters. Figure 9 illustrates how this regulation is expressed.

Instead of defining a simple attribute rule for “Washing Room” with static integer values, now the attribute “Surface Area” references two integer expressions for the lower and upper bound. There is no need to restrict the size of a washing room, therefore the upper bound of the attribute rule is determined by a “STAR” (unlimited). The lower bound is composed of runtime dynamic integer expressions. The “INTEGER_OP_EXPR” is calculated by the multiplication (“op = *”) of the left term, again an “INTEGER_OP_EXPR”, and the right term, an “INTEGER_ATTR_EXPR”. The value of the left term is computed by the product of two “CONST_INTEGER” with value 700. The right one is an integer attribute expression; it gets the number of wash basins which is represented by the actual value of the attribute “NumWashBasins” for the actual washing room.

The above representation (Figure 9, right hand side) shows the internal data structure to store the expression as a graph. The representation below is more readable and thus user friendly; it matches the user interfaces representation.

CONCLUSION

In this paper we introduced an expressive visual knowledge specification language to support conceptual design in civil engineering. Based on graph technology, we described a dynamic knowledge model and a possibility to define design rules. We further presented the expressive power of the visual language with examples. The expressiveness of the three basic design rule types (attribute, relation and cardinality rules) is extended by the concepts of inheritance and aggregation as well as by runtime dynamic expression and complex relations.

The defined knowledge in form of design rules serves on the one hand as reference work for architects and civil engineers. A more powerful usage results from graph-based consistency analyses checking a conceptual sketch. The consistency analyses (Kraft and Nagl 2003), not presented in this paper, are specified in the form of graph transformation using the graph rewriting system PROGRES. The developed tool support (Kraft and Wilhelms 2004) provides a useful representation for knowledge engineers and architects and demonstrate the feasibility of such an approach.

RELATED WORK

In literature there are several approaches to support architects in design. 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. (Gips and Stiny 1972) use shape grammars to formalize knowledge about different classes of buildings, this approach also focuses on the generation of sketches. 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 is introduced. Even if the work is neither implemented nor integrated into a CAD tool, the ideas are fundamental for our research. The semantic web approach (Berners-Lee et al. 2001) aims to improve the quality of information in the World Wide Web. This approach is based on RDF (Powers 2003), a language developed especially for modeling knowledge.

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 how the knowledge should be structured is not explained.

Nosek and Roth were able to show in (Nosek and Roth 1990) that visual semantic networks are easier to understand than textual logic. Visual semantic networks constitute the basis for the visual language presented here. Semantic networks were first introduced in the area of artificial intelligence. Formal concept analysis (Stumme and Wille 2000) and conceptual graphs (Sowa 1984), based on semantic networks, also describe a way to store knowledge in a formally defined but human readable form. The TOSCANA system, which is based on formal concept analysis, describes a tool to store legal building rules. In contrast to our approach, it is restricted to store and classify texts of law, dependencies between laws cannot be represented. Finally, the TOSCANA system is not integrated with a CAD tool.

ACKNOWLEDGEMENTS

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 2005). Furthermore, we would like to thank the Nussbaum GmbH, especially Gerd Schneider, for fruitful discussions and preparing a collection of design rules.

REFERENCES

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

Becker, S. and Westfechtel, B. (2004). „UML-based Definition of Integration Models for Incremental Development Processes in Chemical Engineering”, SDPS, *J. of Integrated Design and Process Science: Transactions of the SDPS*. 8 (1) 49-63. IOS Press.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001). “The Semantic Web”. *Scientific American*, (5). 2001

Böhlen, B., Schleicher, A., Westfechtel, B., and Jäger, D. (2002). “UPGRADE: Building Interactive Tools for Visual Languages”. *6th World Multiconf. on Systemics, Cybernetics and Informatics (SCI2002)*. Information Systems Development I. pp 17-22.

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.

Gips, J. and Stiny, G. (1972). “Shape Grammars and the Generative Specification of Painting and Sculpture”. *Information Processing.* 71:1460–1465.

GRAPHISOFT. (2005). “GRAPHISOFT Homepage”. www.graphisoft.com.

Kraft, B., Meyer, O., and Nagl, M. (2002). “Graph Technology Support For Conceptual Design”. *Proc. of the 9th Intl. Workshop of the Europ. Group for Intelligent Computing in Engineering (EG-ICE2002)*. pp 1-35. VDI Verlag.

Kraft, B. and Nagl, M. (2003). “Semantic Tool Support for Conceptual Design”. *Proc. of the 4th Int. Symp. on Information Technology in Civil Engineering*. pp 1-12. ASCE.

Kraft, B. and Nagl, M. (2004). “Parameterized Specification of Conceptual Design Tools in Civil Engineering”. LNCS 3072, *Proc. of the Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE'03)*. pp 90-105. Springer.

Kraft, B. and Schneider, G. (2005). “Semantic Roomobjects for Conceptual Design Support” *Proc. of CAAD Futures 2005 (accepted)*.

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 (CD-ROM).

Meissner et al. (2005). “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.

Nosek, J. T., and Roth, I. (1990). “A Comparison of Formal Knowledge Representatin Schemes as Communication Tools”. *Intl. J. of Man-Machine Studies*, 33 (2) 227–239.

Powers, S. (2003). *Practical RDF*. O’Reilly.

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

Schürr, A. (1991). *Operationales Spezifizieren mit programmierten Graphersetzungssystemen*. Ph. D. Thesis. RWTH Aachen. Wiesbaden.

Sowa, J. (1984). “Conceptual Structures: Information Processing”. *Mind and Machine*. Addison-Wesley, Boston, MA, USA.

Stumme, G. and Wille, R. (2000). “Formal Concept Analysis on its Way from Mathematics to Computer Science”. LNCS 2393, *Proc. of the 10th Intl. Conf. on Conceptual Structures*. Springer.

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.