ICCCBE-X 2004 - Xth International Conference on Computing in Civil and Building Engineering
ed. K. Beuke, B. Firmenich et al., Seiten 1-14, Bauhaus Universität Weimar, Universitätsverlag (CD-ROM), 2-4 June 2004,Weimar

Interactive distributed knowledge support for conceptual building design

B.Kraft, N.Wilhelms, RWTH Aachen University
{kraft|nilsw}@i3.informatik.rwth-aachen.de

Summary

In our project, we develop new tools for the conceptual design phase. During conceptual design, the coarse functionality and organization of a building is more important than a detailed worked out construction. We identify two roles, first the knowledge engineer who is responsible for knowledge definition and maintenance; second the architect who elaborates the conceptual design. The tool for the knowledge engineer is based on graph technology, it is specified using PROGRES and the UPGRADE framework. The tools for the architect are integrated to the industrial CAD tool ArchiCAD. Consistency between knowledge and conceptual design is ensured by the constraint checker, another extension to ArchiCAD.

1 Introduction

Designing buildings is a difficult task. On the one hand, the architect has to do a creative and artistic work, as the future building should be interesting and attractive. On the other hand, he has to observe a lot of legal, economical, and design-technical restrictions. Moreover, the pure functionality of a building, so the correct arrangement of rooms, their equipment, and meaningful relations between them, have to be guaranteed.

Whereas industrial CAD tools assist the architect during constructive design, the creative and imprecise early design phase, the conceptual design, is not supported. In consequence, the architect has to store all conceptual information in an informal way using simple text, hand-drawings, or just his mind. All restrictions stored in different books, web pages, or just architect's experiences from former building projects have to be regarded in the informal sketch of the future building. Afterwards, the informal conceptual sketch has to be transferred into the constructive design, therefore it has to be manually inserted into a CAD tool. Although CAD tools are powerful and allow comfortable and detailed editing of a sketch, they do not have the capability to store conceptual information, e.g. why rooms are neighbored, or why they are adjusted to the south. This valuable information gets lost, even though it is fundamental for the later design phases.

In the early phase, the specific design of a future building is less important, than the complete and precise information management. To understand the requirements and restrictions of a building project, to discuss them with the investor, and finally to fix this information in an adequate semi-formal notation are the main tasks of this phase. Based on this information, the functionality and structural organization of a building can be elaborated. Functional areas are identified and related to each other, room types are defined and linked. The result is a draft of the building in an abstract form. We call this first phase the conceptual design. As dimensions and positions of rooms are less important for the conceptual design, this information is not represented. Actually there exists no adequate industrial tool support for this phase. The complete and precise definition of requirements and restrictions are crucial for the correctness of the future building. Misunderstandings and missing information lead to design errors that are propagated to all following design phases. As errors made in early design phases are later difficult and expensive to repair, the correctness of the conceptual design phase is essential for the success of a building project.

2 Using Graphs for Conceptual Design Support

Graphs are a powerful and flexible data structure to model problems from different application domains. We develop graph based support for the domain of architectural engineering, especially for the conceptual design (Kraft and Nagl 2003; Kraft, Meyer et al. 2002). Developing a graph structure means identifying node types to represent important entities, and to define edge types to express relations between them.

Currently, CAD tools just provide constructive design elements, e. g. walls that are not capable to express relevant conceptual information. Therefore we introduce new design elements, which are more intuitive to use and capable to store conceptual information. Such elements are rooms and areas. Rooms are in our scenario the most important entities, as each building basically consists of rooms. Areas describe an aggregation of several rooms. Both are represented by the node class semantic object. To further describe properties of these entities, we provide a node class attribute, which is assigned to semantic objects. The node class relation allows defining interrelationships between semantic objects. In chapter 3 the graph schema is described in detail.

We use the graph rewriting system PROGRES to specify and execute graph transformations. PROGRES (PROgrammed Graph REwriting Systems) is a very high level, operational specification programming language for rapid prototyping (Schürr 1991). In our architecture engineering project we use graph technology for two purposes. First, we provide methods to edit the graph, to build and modify a visual data structure for knowledge definition. Furthermore, we use graph transformations to check the sketch and to inform the architect in case of inconsistencies. We search for inconsistent sub graphs and create error messages if such an inconsistency is found. The error messages, of course, are also represented by a graph node.

Currently, CAD tools allow modeling sketches and give a broad support for the construction, but the concept of a building cannot be modeled. Moreover, there exists no tool support to analyze the sketch and to check it against legal, economical, or technical restrictions.

In our idea, conceptual design support therefore consists of two main parts. First, the domain specific knowledge has to be formalized for each class of buildings by a knowledge engineer. We suppose that the knowledge engineer is usually an experienced architect, without any programming capabilities. Therefore, we develop a visual language for knowledge definition, and a graph based application implementing this language. Second, existing CAD tools are extended to allow architects to design in a conceptual way. These extensions are easy to use and promote the architect's creativity. Both parts are integrated, so that the architect's sketch can be checked against the defined knowledge.

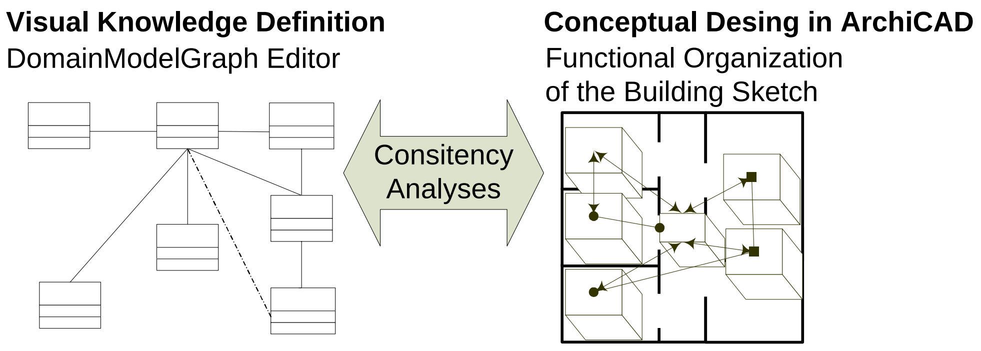

Figure 1: Complete Scenario

Figure 1 depicts the complete scenario of our project . The *Domain Knowledge Graph Editor*, on the left side, provides functionality to define conceptual knowledge. The editor is developed using graph technology. Based on a fixed graph schema, it allows to dynamically define semantic objects in a knowledge model, and to formalize rules, specific for a class of buildings. Our goal is to give the knowledge engineer a tool for visual knowledge definition, which he can use without having any programming experience. The Domain Knowledge Graph Editor represents knowledge in a formal, but human readable and easy to understand form.

To support the conceptual building design, in Figure 1 depicted on the right side, we extend the CAD tool ArchiCAD with new functionality (Kraft 2003). Our main extensions to ArchiCAD are *roomobjects*. These new design elements represent a room's functionality. Using roomobjects, the architect can explicitly design the conceptual and functional organization of a building. Roomobjects support a more intuitive workflow; conceptual information, which usually gets lost, is preserved in the sketch.

Based on the formalization of knowledge and design, the sketch can be checked. Currently we export the knowledge elaborated with the Domain Knowledge Graph Editor into a textual file using the Resource Description Framework (RDF) (Powers 2003). The RDF file is interpreted by the *constraint checker* in ArchiCAD. It checks the architect's sketch and informs him in case of restriction violations. The correction is not done automatically. The architect is free to fix the error or to keep in an inconsistent state. Thus the creativity and design freedom are not restricted.

In the following two chapters, we will first describe the tool for knowledge definition and its system design. Then, our ArchiCAD extensions and the consistency analysis are explained. At the end of chapter 4 we present a way to reach control integration between both parts in a distributed scenario.

3 Visual Knowledge Definition for Conceptual Design

In our approach, we follow a visual, graph-based knowledge representation similar to the semantic web. The main advantage of this approach is the clarity for humans and processibility for computers. We use the PROGRES system to specify a graph schema and graph transformations that allow dynamically developing a knowledge model, representing the relevant entities for a specific class of buildings. These entities constitute the basis for actual knowledge definition, e.g. legal, economical, or technical restrictions. Finally, we develop a graphical user interface, optimized for a clear arrangement and intuitive definition of the knowledge.

The benefits of this formalization process are on the one hand the preservation of knowledge, this is especially essential for experience values which are usually not explicitly stored. The most important benefit comes on the other hand with graph-based consistency analyses that check the architect's sketch against the formalized knowledge. Thus, design errors are identified as early as possible.

3.1 Graph Schema

The graph schema for conceptual knowledge representation is depicted in the upper part of Figure 2. The schema is fixed in the PROGRES specification; it defines the syntax and expressiveness of the visual language for conceptual knowledge representation. As it is specified in PROGRES, it cannot be changed at runtime. We distinguish between three basic concepts, namely *semantic object*, *relation*, and *attribute*, each represented by a PROGRES node class. These basic concepts and their associations restrict the general data structure graph to a subclass specialized for knowledge definition in the domain of conceptual design. This subclass, however, is still general enough to cover knowledge definition for any class of buildings.

The node class semantic object, depicted in the middle of the graph schema, stands for the basic elements, a building is made of. As our goal is to model conceptual design information, these elements represent functional elements like rooms or areas, in contrast to constructive elements like walls or columns. Two subclasses *complex* and *atom* inherit from the node class semantic object. An atom is indivisible and describes the smallest possible entity in the knowledge definition. Several atoms can be combined to a common unit, represented by the node class complex. They can again be combined to bigger units and so on. E. g. in a building, each floor consists of several rooms, the building itself consists of several floors. In the graph schema, the aggregation is represented by a *contains* edge from the node class complex to atom, and the node class complex to itself respectively. Whereas the concept of the node class complex is usually to aggregate units heterogeneously, the concept of classification models homogeneous structures. Classification is realized by the inheritance relation, represented in the graph schema as an *isA* edge.

The node class *attribute* serves to describe properties of semantic objects. As the definition of specific properties should be possible at runtime, the graph schema only provides general data types for a property definition. *Boolean* attributes e. g. can define a certain equipment to be necessary or forbidden. *Range* attributes prescribe certain values to be inside an interval, they are again specialized to integer and real range restrictions. *Enum* attributes allow defining a set of strings.

Two semantic objects can be interrelated using the node class *relation*. Again, this node class does not express a specific relation like access between two entities, but only the general, unspecific relation. Analogously to the attribute definition, the knowledge engineer is responsible for defining each relation and its semantics in the knowledge model at runtime. The graph

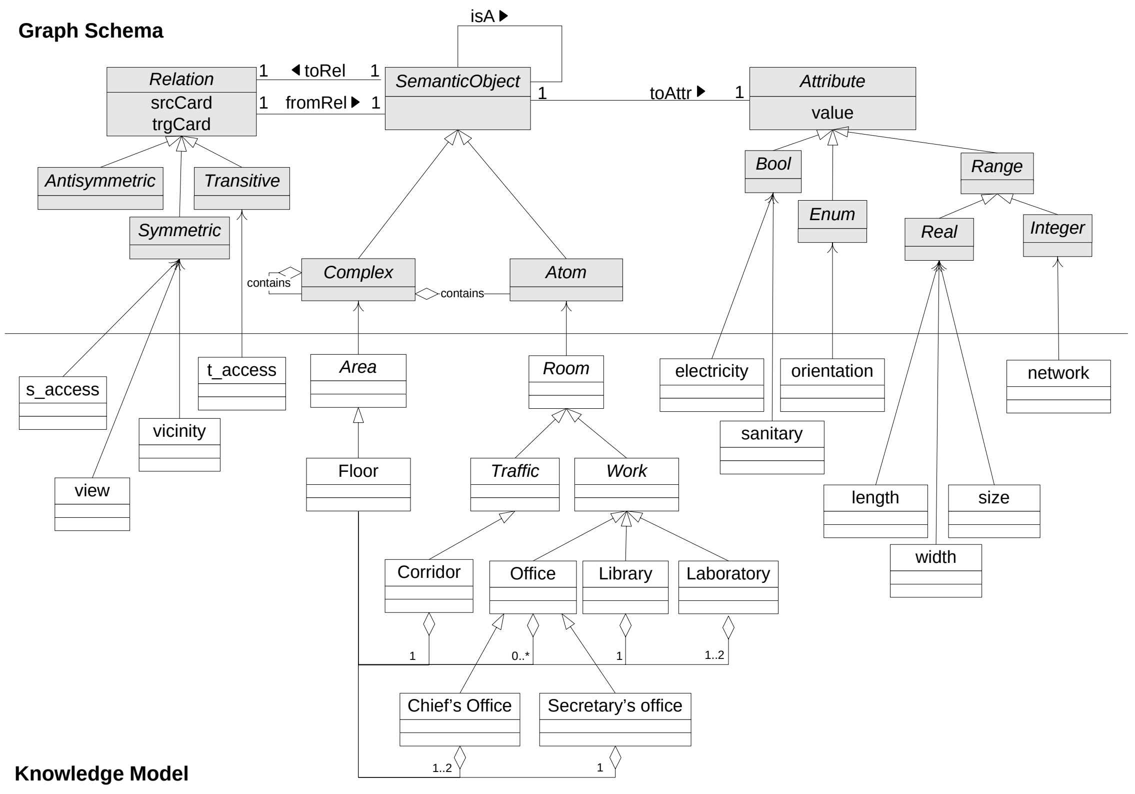

Figure 2: Graph Schema and Knowledge Model

schema allows defining forbidden and obligatory relations through cardinality restrictions. If e. g. the direct access between the corridor and chief's office should be forbidden, the value of the node attributes *source* and *target cardinality* is set to zero. Any value different from zero expresses an obligatory relation. The cardinality attributes restrict the minimal and maximal number of connected semantic objects. The node class relation is specialized to *antisymmetric*, *symmetric*, and *transitive* relations. The access relations e. g. is usually symmetric, but can also be restricted to one direction. Moreover, the transitive access relation expresses a general accessibility between semantic objects.

3.2 Knowledge Model Definition

The graph schema described in the previous section constitutes the basis for the knowledge model definition. In contrast to the fixed graph schema, the knowledge model is elaborated dynamically by the knowledge engineer. So, he is able to flexibly create a knowledge model which is optimized for the building type specific needs; especially the level of abstraction is not fixed. This kind of flexibility is essential, because each knowledge model references a specific class of buildings. Operations on the knowledge model and consistency analyses, especially the graph transformations, are parameterized, too, to guarantee this kind of flexibility (Kraft and Nagl 2004).

In the lower part of Figure 2, an example model for the class of office buildings is shown. For readability reasons the same representation for inheritance and aggregation relations is used as in the graph schema, though the semantics is slightly different. For the same reason, the graph schema is not represented as a UML Meta model, but in a simplified form.

Looking at the knowledge model in Figure 2, the class *room* instantiates the node class atom. Therefore, it represents the basic and smallest functional element of the knowledge definition for an office building. The class room itself is specialized into the classes *work*, representing working rooms, and *traffic*, representing a super class for all rooms where people usually don't stay. The specialization of room classes gives an overview of the functional decomposition of the future building. E. g. the *corridor* is a special traffic room. Analogously, the *chief's office* is an extension to a general *office*. Again, the level of specialization and thus the level of detail are not restricted; it can be elaborated dynamically at runtime. Knowledge can on the one hand be defined using the leafs of the classification tree. These rules are valid for all instances of the corresponding semantic objects. On the other hand, knowledge defined for super classes, e. g. for the room class work, is valid for all inheriting classes. We further distinguish between abstract and concrete classes. Concrete classes form the basis for conceptual building design, each building is a composition of their instances. Abstract classes cannot be used for design. In the knowledge definition, however, concrete and abstract classes can be instantiated as the knowledge is inherited by the subclasses.

The node class complex is used to model the aggregation of several entities to a common unit. Complex classes allow defining knowledge not only about a single entity, but knowledge concerning several units in a certain context. Here, the class *floor* allows describing knowledge valid for the whole floor considering its internal structure. Moreover, the number of rooms of each class on a floor can be restricted to precise the structural information of this aggregation, in Figure 2 indicated by cardinalities at the aggregation edges.

Classification and aggregation of semantic objects determine a functional decomposition of the main entities. To specify further properties of semantic objects, the knowledge engineer defines attribute classes. The attribute class *size* e. g. determines the minimal and maximal size of a semantic object as an interval of valid floating-point numbers. In the knowledge model depicted in

Figure 2, the attribute classes electricity and sanitary are defined as Boolean. They allow specifying the obligation or prohibition of the installation of the corresponding equipment.

The definition of relation classes allows specifying knowledge about interrelationships between semantic objects. Using e. g. the *s_access* relation one can describe the obligation or prohibition to have a direct, symmetric access between two semantic objects in an actual building. A connection of class *t_access* models the transitive, indirect accessibility. Again, the high flexibility of our knowledge model enables the knowledge engineer to dynamically elaborate relation concepts at runtime, also those like *vicinity* that are not as obvious as access.

3.3 Conceptual Knowledge Definition

Up to here, we have described the graph schema, which restricts the general data structure graph to the needs for knowledge modeling in civil engineering. Based on the schema, the knowledge model definition can dynamically be elaborated. Each knowledge model is specific for one class of buildings. The knowledge model again constitutes the basis for the conceptual knowledge definition. The resulting set of rules records knowledge valid for all buildings of the corresponding class, but not only for one actual building. Because of the consequential reusability, the effort of defining the knowledge base pays off.

According to the graph schema, we distinguish between *attribute rules* and *relation rules*. Attribute rules describe the restriction of attribute values for a semantic object, relation rules prescribe the characteristics of the relationship between two semantic objects. Furthermore, we present a complex relation rule using generalization, a path expression, and derived cardinalities.

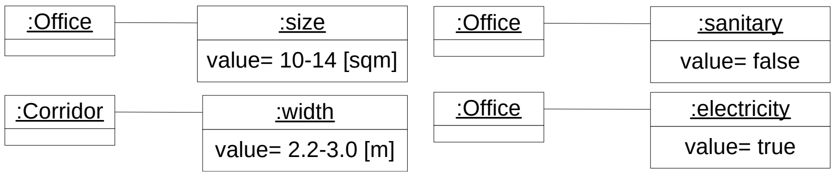

Figure 3: Attribute Rules

Figure 3 depicts attribute rule examples for instances of the office and the corridor class. The first attribute rule concerning the office denotes a size restriction. That means that the dimension of each office in an actual building of the corresponding building class has to be at least 10 and at most 14 square meters. Analogously, the width of all corridors is restricted to be inside the specified interval of 2.2-3.0 meters. Boolean attribute rules denote the obligation or prohibition of certain equipment to be installed. E. g., each office should have electricity installation but none should have sanitary installation. Rules on the knowledge level concern all occurrences of the corresponding semantic object in an actual building. The rules describe knowledge on the type level.

Examples for relation rules are depicted in Figure 4. The top most relation rule demands each secretary’s office to be near by a printer room. To refine this statement, cardinality attributes of the relation restrict the number of the connected semantic objects. The source cardinality refers to the left hand side of the relation, the target cardinality to the right one. In this case, the secretary’s office has to be near by at least one printer room. Moreover, each printer room can only be appropriate for at most three secretaries, to restrict the expected printing amount. Analogously, one secretary can serve at most two chiefs, represented by the direct access. On the other hand, each chief’s office has direct access to exactly one secretary’s office. Thus, each chief’s office has an indirect access to the corridor, the direct access to the corridor is not required. In Figure 4 the direct access is actually forbidden to avoid disturbances. The prohibition

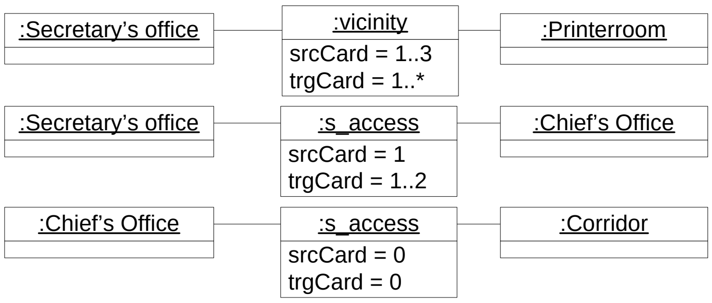

Figure 4: Relation Rules

of a relation is modeled by cardinality restrictions equal to zero at both relation ends. Again, the knowledge is defined on the type level, all rooms in the future building must fulfill all rules.

Figure 5 shows a more expressive relation rule. Like above, it consists of two semantic objects connected by a relation with cardinality restrictions. The expression power of this rule grows using the inheritance hierarchy, transitivity, and a derived cardinality. The source part of the relation rule represents a semantic object on a high level in the class hierarchy. Thus, the relation rule is valid for all inheriting classes. As the class *room* is even a root class of the knowledge model in Figure 2, the depicted rule concerns all atom classes in the knowledge definition. As the relation *t_access* is an instance of the node class transitive relation, the rule demands the transitive access between any room and the exit. To ensure, that absolutely each room is transitively accessible from the exit, the source cardinality of the relation is equal to the actual number of rooms in future building. The expression *card* dynamically calculates the number of occurrences of certain semantic objects. The rule is parameterized depending on the actual building sketch.

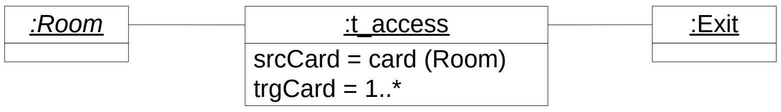

Figure 5: Generalized, transitive Relation Rule

Using attribute and relation rules, only restrictions to the future building, but no design proposals are defined. Therefore, everything not specified is optional. If e. g. no size restriction has been defined for a semantic object, the corresponding entity's size in the building sketch can be arbitrarily set by the architect. Likewise, the architect can sketch e. g. an access between two rooms, if no relation rule exists.

3.4 Tool Support for Visual Knowledge Definition

Up to here, we presented a visual language for conceptual knowledge definition. To provide a tool support for the knowledge engineer, we develop a graph based prototype, called the Domain Knowledge Graph Editor depicted in Figure 6. This application serves for elaborating, modifying, and administrating knowledge for the domain of civil engineering. To provide a compact and clearly arranged view on the knowledge definition, instances of semantic objects are displayed in a UML class diagram (Fowler and Scott K. 1999) similar representation. Even if the depicted graph is rather small, it already contains over 70 design rules.

In our department we develop graph based tools using the PROGRES language (Schürr 1991) and the UPGRADE framework (Böhlen, Schleicher et al. 2002). The PROGRES system (Winter 2000) provides comfortable visual programming; it incrementally checks the syntax and static semantics of the specification. Furthermore, the PROGRES system allows generating efficient C-code, which is used to produce so called UPGRADE prototypes. UPGRADE is an acronym for Universal Platform for GRAph-based Development, it is a framework to develop tools for visual languages like PROGRES. UPGRADE automatically generates an appropriate prototype from the given C-code which offers the user an adequate graphical interface. The user can then work on the host graph. He can execute all PROGRES transformations to create and modify the current graph. The tool developer can adapt the appearance of the prototype to the needs of the application domain. The prototype additionally allows defining filters, to emphasize certain node types or to hide them. The PROGRES specification for the domain knowledge graph is described in (Kraft and Nagl 2004). The Domain Knowledge Graph Editor, depicted in Figure 6, is a result of this tool construction process.

The Domain Knowledge Graph Editor is the tool for conceptual knowledge definition. The knowledge engineer uses it, to define conceptual knowledge specific for a class of buildings. On the left hand side of the Domain Knowledge Graph Editor, there are three tree views, containing the classes of the knowledge model. In the topmost tree view the room classes are presented, e. g. the room class office. Up to now only atom classes are supported. Next to each class, an icon is displayed symbolizing the semantics of a room. The tree view beneath contains attribute classes, the last tree view represents relation classes. Again, each class is associated with a descriptive icon.

The main part of the Domain Knowledge Graph Editor is called the *graph view* containing the domain knowledge graph. In Figure 6 a basic domain knowledge graph for office buildings is already defined. In this view, all attribute rules of one semantic object are arranged in one box. The icons of the corresponding classes are displayed, too. Next to the name of each attribute rule, its value and unity are displayed. E. g. the length attribute of all conference rooms should be within the interval 500 and 1200 cm. All seven further attribute rules for the conference room are arranged in the same box, instead of displaying each rule separately. A layout algorithm provides this clear and compact representation.

Edges between semantic objects represent instances of a relation class. The cardinality restrictions of each relation are displayed at the edge's ends. E. g. an access relation is defined between the classes corridor and office. In an actual building, each corridor must have access to an arbitrary number (0..*) of office rooms in this building. Each office, however, must be accessi-

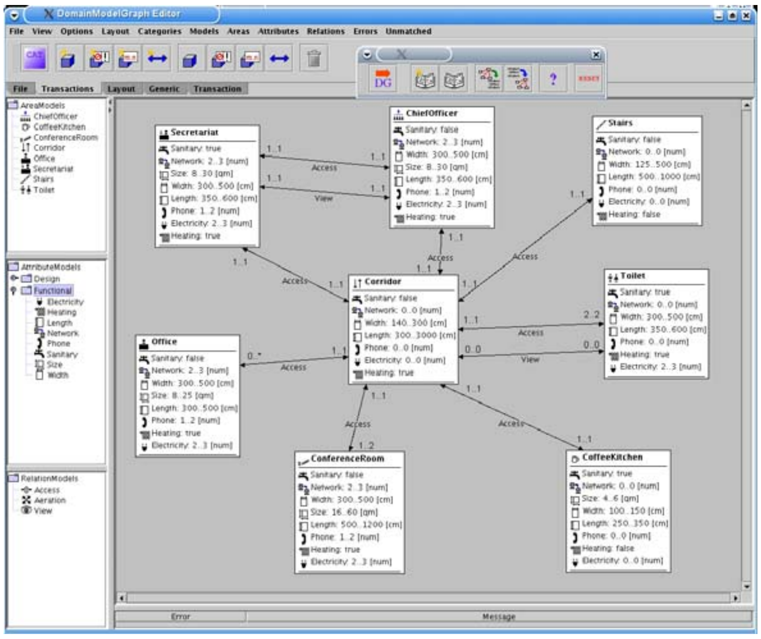

Figure 6: Domain Knowledge Graph Editor

ble from exactly one (1..1) corridor. Relationships can also be forbidden: There should be no visibility between toilets and the corridor, in the example this rule is indicated by a view edge with the cardinality restriction (0..0).

To ensure, that the domain knowledge graph does not contain internal inconsistencies and contradictions, our knowledge definition tool provides a so called internal consistency analysis. E. g. a coexistence of an obligatory and a forbidden relation of the same class between the same two semantic objects would be recognized by the tool; the knowledge engineer would get a message for this inconsistency.

The knowledge defined with the aid of the Domain Knowledge Graph Editor can be used to analyze a given sketch. In order to do so, one can export the defined knowledge. There are two different export formats provided. The first one is based on the Graph eXchange Language (GXL) (Winter, Kullbach et al. 2002) format, which is a generic XML file format for graphs. This export is used for backup and restore within the tool. An eXtensible Stylesheet Language (XSL) transformation (w3.org 2004) automatically generates a HTML documentation of the defined knowledge, containing further information about the exported graph. The second export format is based on the Resource Description Framework (RDF) (Powers 2003). An exported RDF file can be imported into ArchiCAD and processed to analyze the architect's sketch.

4 Conceptual Design in ArchiCAD

The knowledge definition described in the previous chapter is integrated to support conceptual design. We extend the CAD tool ArchiCAD with new functionality in two ways. First, we give the architect the possibility to use ArchiCAD for the conceptual design phase. Second, we provide consistency analyses that check the sketch in ArchiCAD against the defined knowledge.

In the early design phases, size and arrangement of rooms in the future building are not yet fixed, it is even the main task to elaborate. The main construction element in ArchiCAD is the wall, which is not applicable for this design phase. To enable the architect to design conceptually using ArchiCAD, we extend the ArchiCAD product model with *roomobjects*. In the two-dimensional representation of a roomobject, the room type and actual size information are displayed, the three dimensional representation serves for estimating the volume. In Figure 7, four example roomobjects are depicted. They are based on the knowledge model, described in section 3.2. For each concrete class of the knowledge model, we generate a roomobject and make it available to the architect. Roomobjects are based on the Geometric Description Language (GDL) (GRAPHISOFT 2004), which is provided by GRAPHISOFT to create additional design elements. Thus, the architect can use roomobjects in ArchiCAD in the same way he uses all other construction elements, e. g. walls or columns. He does not need to learn a new handling.

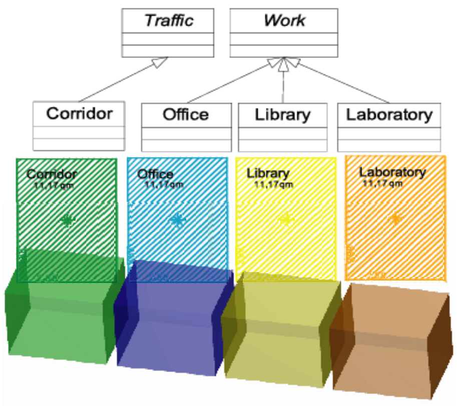

Figure 7: Roomobject – Conceptual Design Extensions to ArchiCAD

Figure 8 depicts an example sketch elaborated in ArchiCAD using roomobjects and roomlinks. Roomlinks allow modeling the concept of the relations described in chapter 3. They serve for defining relationships between roomobjects. The sketch in Figure 8 models the conceptual design of an office floor. The cutout of the floor depicts the chief's office, with access and view to the secretary's office, some standard offices accessible from the corridor and a printer room. Sketching with roomobjects and roomlinks allow a creative and flexible elaborating of alternatives without being restricted by a wall structure. In a final step, the conceptual design in ArchiCAD can automatically be transferred into a traditional wall construction, using the so called *wall generator*.

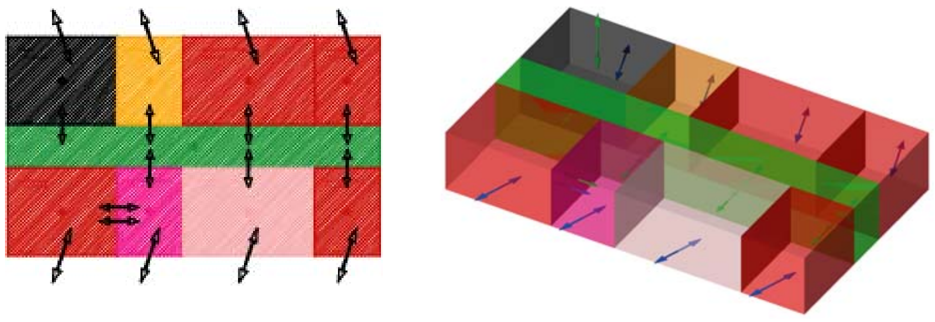

Figure 8: Conceptual Design in ArchiCAD

4.1 Consistency Analyses

Existing CAD tools do neither provide the definition and processing of knowledge, nor to sketch conceptually. Thus, there exists no checking of the architect's sketch, especially during the early design phases. In our approach, we provide both, the definition of knowledge using graph based tools and the conceptual design by extending ArchiCAD. The already described benefits comprise the explicit storage of rules and experiences, and the preservation of the conceptual information during the early design phase. The main advantage comes with the possibility of using the knowledge to check the sketch. The formal specification of knowledge and conceptual design allows defining consistency analyses.

The constraint checker, another extension to ArchiCAD, can process defined knowledge and analyze the sketch in order to find rule violations. The constraint checker is integrated in ArchiCAD using a programming interface (C-API 5.1), provided by GRAPHISOFT. The constraint checker can be run at any time during the conceptual design. Error messages are displayed next to the corresponding roomobject; they contain a description of the violation and a reference. The architect can concentrate on the creative work as he does not need to remember all valid restrictions.

Currently, we support two different ways to transfer the knowledge into ArchiCAD. On the one hand, we provide import functionality based on the RDF export file, described in section 3.4. The file is interpreted and processed by the constraint checker. In this case the main part of the consistency analyses is done inside of ArchiCAD. Each roomlink and each roomobject is inspected, room positions and size restrictions are examined. On the other hand we plan to provide distributed consistency analyses using CORBA. In this case, the consistency analyses are done by graph based tools, which allow more powerful analyses.

4.2 Distribution Scenario

Knowledge for civil engineering is not static. Many restrictions often change or become obsolete, new restrictions are added, especially legal restriction are affected. Therefore, a locally installed knowledge base, like it is described above, would need to be updated frequently. We suppose a central knowledge server providing the latest version of the knowledge. In this scenario one knowledge engineer is responsible for the maintenance, many architects can access the knowledge server to check their sketches.

Instead of the local constraint checker, described in section 4.1, a central, graph based constraint checker would execute the analyses. A constraint checker based on graph technology using PROGRES allows complex graph pattern matches that offer powerful analyses for conceptual design. It would be difficult to fully implement graph technology support in ArchiCAD. Moreover the effort implementing a complete constraint checker would be necessary for each CAD tool. Therefore we exemplarily integrate ArchiCAD with the central consistency analysis, running in a central UPGRADE prototype. The constraint checker in ArchiCAD is then replaced by a simple communication interface which only serves for transmitting and receiving data. We implement a CORBA (Puder and Römer 2000) interface for both applications to establish the communication between them using the internet. Figure 9 depicts this distribution scenario.

Using the integration of knowledge and design, each operation performed by the architect, is transmitted to the central graph based consistency checker. For each architect connected to the graph tool, an own shadow graph is build up a as basis for the graph based analyses. The shadow graph, we call it the *design graph*, stores the architect's sketch in an abstract graph representation. Inconsistencies are identified in the design graph using PROGRES graph transformations and tests. Inconsistent sub graphs are matched, a corresponding error is transmitted to ArchiCAD where an error message is visualized.

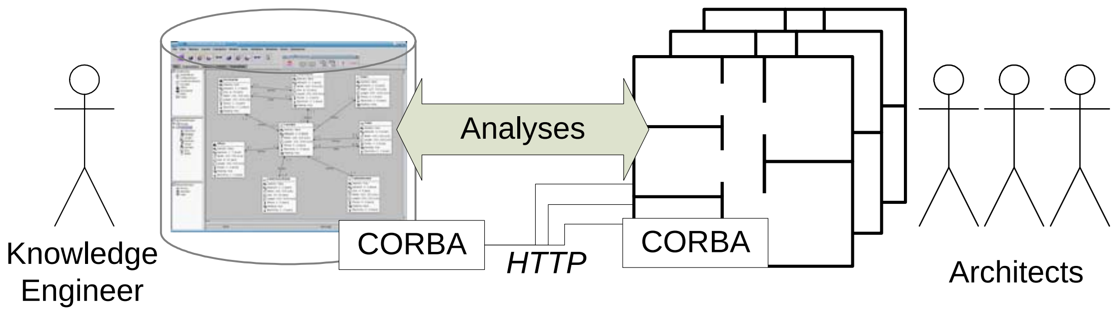

Figure 9: Control- Integration between Knowledge and Design

5 Summary and Related Work

In this paper we introduced a knowledge support for conceptual design in civil engineering. Based on graph technology, we described a fixed graph schema, a dynamic knowledge model, and a possibility to define knowledge on the type level, specific for a class of buildings. We further presented an application for knowledge definition and evaluation. To enable the architect to design conceptually in the CAD tool ArchiCAD, we introduced roomobjects and roomlinks. These new design elements allow elaborating a sketch in the early design phase using ArchiCAD. Consistency between the sketch in ArchiCAD and the defined knowledge can be checked in two ways. First, the knowledge definition can be exported to a RDF file, second, we provide a control integration of our graph-based tools with ArchiCAD.

There are several approaches to support architects in design. Christopher Alexander describes a way to define architectural design patterns (Alexander 1995). Although design patterns are extensively used in computer sciences, in architectural design this approach has never been formalized, implemented, and used. Graph rewriting has been used by (Göttler, Günther et al. 1990), to build a CAD tool that supports the design process of a kitchen. In (Borkowski and Grabska 1998; Borkowski, Schürr et al. 2002) graph grammars are used to find optimal positions of rooms and to generate an initial floor plan as a suggestion for the architect. In (Szuba and Schürr 2004) a graph based support for modeling process knowledge using the PROGRES system is described. In contrast to our approach, the knowledge is hard wired in the specification and cannot be elaborated at runtime. The SEED (Flemming 1994) system provides a support for the early phase in architectural building design. The different modules, SEED-Pro, SEED-Layout and SEED-Config allow specifying requirements of the buildings, generating floor plans and three dimensional models based on these requirements. However, it does not provide an interactive, integrated tool support. The importance of knowledge processing for architectural design is comprehensively discussed in (Coyne, Rosenman et al. 1990; Gero 1999).

(Steinmann 1997) and (Heck 1998) describe models and data structures for a new, intelligent CAD tool. Analogously to our approach, Steinmann introduces classification and aggregation relation. In contrast to our approach, the expression power is restricted to the classification and to attribute evaluation. Moreover, the integration to an existing CAD tool is only used for transferring design. In our approach we integrate conceptual knowledge with conceptual design inside the CAD tool.

Knowledge representation based on semantic web is described in (Gomez-Perez, Fernandez-Lopez Mariano et al. 2004). The Resource Description Framework defines a language for ontology and knowledge definition (Powers 2003). Even if a lot of ontologies have already been developed, none of them is applicable for the conceptual design phase. On the insufficiency of ontologies report (Silva, Vasconcelos et al. 2002), they present alternative solutions, unfortunately none in the field of graph grammars. Object oriented knowledge representation approaches based on UML (Fowler and Scott K. 1999) and OCL (Clark and Warmer 2002) is described in (Martin 2003).

Formal concept analysis (Stumme and Wille 2000) and conceptual graphs (Sowa 1984) 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.

Acknowledgement

The authors gratefully acknowledge the support of this project by the German Science Foundation (DFG) within the scope of the priority program “Network-based Co-operative Planning Processes In Structural Engineering” (SPP 1103).

References

Alexander, C. (1995). A Pattern Language. Löcker.

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

Borkowski, A. and Grabska, E. (1998). Converting Function into Object. In Artificial Intelligence in Strutural Engineering. LNAI 1454. ed. I. Smith. pp 434-439. Springer.

Borkowski, A., Schürr, A., and Szuba, J. (2002). GraCAD - Graph-Based Tool for Conceptual Design. In Proc. of the 1st Int. Conference on Graph Transformation (ICGT2002). LNCS 2505. pp 363-377. Springer.

Clark, T., and Warmer, J. (2002). Object Modeling with the OCL - The Rationale behind the Object Constraint Language. Springer.

Coyne, R. D., Rosenman, M. A., Radford, A. D. et. al. (1990). Knowledge Based Design Systems. Addison-Wesley.

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

Fowler, M., and Scott K. (1999). UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley.

Gero, J. S. (1999). Recent Design Science Research: Constructive Memory in Design Thinking. In Architectural Science Review 42. pp 3-5.

Gomez-Perez, A., Fernandez-Lopez Mariano, and Corcho Oscar (2004). Ontological Engineering. Spinger. London.

Göttler, H., Günther, J., and Nieskens, G. (1990). Use Graph Grammars to Design CAD-Systems. In Graph Grammars and Their Application to Computer Science. LNCS 291. eds. G. Rozenberg, M. Nagl, and A. Rozenfeld. pp 396-409. Springer.

GRAPHISOFT (2004). GDL Homepage. www.gdlcentral.com.

Heck, P. (1998). Ein objektorientiertes CAD-Modell für die raum- und bauteilorientierte Bearbeitung von Gebäuden in der Vorplanung. Ph. D. Thesis. TU Kaiserslautern.

Kraft, B. and Nagl, M. (2003). Semantic Tool Support for Conceptual Design. In Proceedings of the 4th Int. Symposium on Information Technology in Civil Engineering. ed. I. Flood. pp 1-12. ASCE (CD-ROM).

Kraft, B. and Nagl, M. (2004). Parameterized Specification of Conceptual Design Tools in Civil Engineering. In Proc. of the Int. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE'03). LNCS 3072. eds. J. Pfalz, M. Nagl, and B. Böhlen. pp 85-100. Springer.

Kraft, B. (2003). Conceptual Design mit ArchiCAD 8. In GRAPHISOFT.NEWS. GRAPHISOFT Deutschland.

Kraft, B., Meyer, O., and Nagl, M. (2002). Graph Technology Support For Conceptual Design In Civil Engineering. In Proc. of the 9th Int. Workshop of the Europ. Group for Intelligent Computing in Engineering (EG-ICE2002). eds. M. Schnellenbach-Held and H. Denk. pp 1-35. VDI Verlag.

Martin, P. (2003). Plans to extend UML for knowledge representation. http://meganesia.int.gu.edu.au/~phmartin/WebKB/doc/model/comparisons.htm.

Powers, S. (2003). Practical RDF. O'Reilly.

Puder, A., and Römer, K. (2000). MICO: An Open Source CORBA Implementation. Morgan Kaufmann.

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

Silva, F., Vasconcelos, V., and Robertson, D. (2002). On the Insufficiency of Ontologies: Problems in Knowledge Sharing and Alternative Solutions. In Knowledge Based Systems. pp 147-167.

Sowa, J. (1984). Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley.

Steinmann, F. (1997). Modellbildung und computergestütztes Modellieren in frühen Phasen des architektonischen Entwurfs. Ph. D. Thesis. Weimar.

Stumme, G., and Wille, R. (2000). Begriffliche Wissensverarbeitung. Springer. Heidelberg.

Szuba, J. and Schürr, A. (2004). On Graphs in Conceptual Engineering Design. In Proc. of the Int. Workshop on Application of Graph Transformation with Industrial Relevance (AGTIVE'03). LNCS 3062. eds. J. Pfaltz, M. Nagl, and B. Böhlen. pp 71-85. Springer.

w3.org (2004). XSL Transformations (XSLT). www.w3.org/TR/xslt.

Winter, A., Kullbach, B., and Riediger, V. (2002). An Overview of the GXL Graph Exchange Language. In Software Visualization, State-of-the-Art Survey. LNCS 2269. ed. S. Diehl. pp 324-336. Springer.

Winter, A. J. (2000). Visuelles Programmieren mit Graphersetzungssystemen. Dissertation. RWTH Aachen. Aachen, Germany.