ITCE-2003 - 4th Joint Symposium on Information Technology in Civil Engineering
ed Flood, I., Seite 1-12, ASCE (CD-ROM), Nashville, USA

Semantic Tool Support for Conceptual Design

Bodo Kraft, Manfred Nagl

Department of Computer Science III, RWTH Aachen, D-52074 Aachen
PH +49/241/8021314 {kraft|nagl}@i3.informatik.rwth-aachen.de

Introduction

When designing a building, an experienced architect implicitly applies his aggregated knowledge to the new sketch. In this early design phase, called conceptual design, most architects do not elaborate their sketches using a CAD system. They rather work with pencil and paper. Without being directly aware of, the architect considers design rules, functional requirements, economic and legal restrictions. Constructive elements, like walls, windows or doors are used with their conceptual meaning, namely to form organizational areas or rooms, to guarantee e.g. light and ventilation, or to ensure accessibility. These conceptual elements, therefore, form a functional view of the design structure which, however, is not explicitly defined. Existing CAD systems give no support for this creative conceptual design.
There is no smooth transition to constructive design. The architect manually elaborates the constructive design, now using a CAD system. He replaces the functional elements of the sketch by constructive ones, e.g. ventilation by a window, the access by a door etc. The conceptual information he had in mind gets lost. Furthermore, there are many changes within the development process. E.g., if the client is not satisfied, the architect has to go back to the conceptual design. The modified conceptual data are lost again after the next transformation step. Such iterations are risky and expensive in terms of time and money.

Conceptual Design in Civil Engineering

We divide the building construction process into three main phases. Figure 1 depicts these different design phases, the level of details increases from a very abstract view on the whole building in the conceptual design, over the construction of a building using CAD systems in the constructive design, to the calculation and dimension of special aspects like heating and cooling system or static calculations. Currently there is no integration between these phases, the defined concepts have to be manually transferred. The conceptual design phase is currently not supported by any tool. Moreover, neither in the constructive design, nor in the detail design phase the application of conceptual knowledge is supported. The lack of representation possibilities of the conceptual design information forces the architect to keep the information in mind, in fact a lot of them get lost.

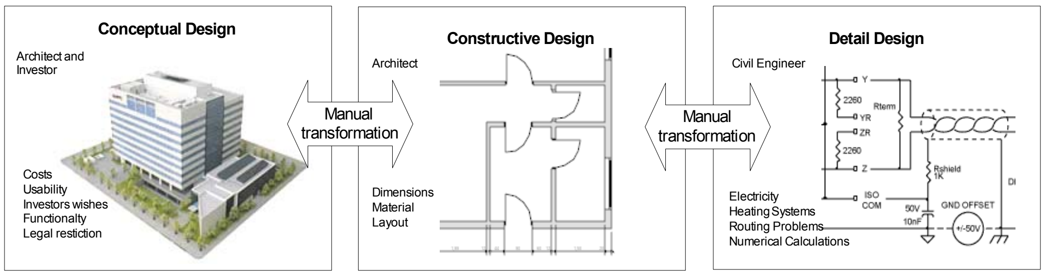

Figure 1: Phases of the architectural design process

During conceptual design the functionality and requirements of a building are fixed by an architect and an investor. Important aspects of the conceptual design are the building type, room types, room relations, floor space, the functional connections and early cost calculations. The investor is more interested in the usable area and room plans than in detailed material definitions or in structural analysis. The result of this phase is a coarse and informal sketch of the building and a textual definition of the requirements.
To plan e.g. an office block, first the number and size of stories has to be fixed. Further, the usable areas, especially the offices are planned. The number and size of offices depends on the wishes of the investor, technical and legal restrictions. Depending on the usable area the traffic area and sanitary installations can be dimensioned and defined. All this can be fixed in this early phase without considering exact positions and room arrangements. Currently CAD systems do not support the definition of requirements or a functional view of a building.

Complete Scenario

Our research to bridge conceptual and constructive design is organized in two approaches, which differ on the one hand in the used methodology, on the other hand in the way to support the architect. In the top-down approach (see left side of Figure 2), we build tools for elaborating conceptual knowledge by a knowledge engineer. Further tools for conceptual design support an architect and incrementally check his conceptual design results against the available conceptual knowledge. Our goal is to find conceptual errors as soon as possible. The top-down tools serve as an experimentation platform, to find suitable concepts and to evaluate their acceptance. This approach, especially, pays off if the conceptual knowledge is restricted to specific classes of buildings, and if several conceptual designs within a class are worked out.
In the bottom-up approach, which is more industrially oriented, we extend the CAD tool ArchiCAD [GRAPHISOFT1999] by introducing conceptual objects, such as room types or areas built up by rooms and by defining corresponding relations. Again, a constructive design result received by using ArchiCAD is checked versus the underlying conceptual knowledge. To allow also for traditional design processes, a layout generator creates an initial floor plan from the conceptual design result as a starting point for constructive design.

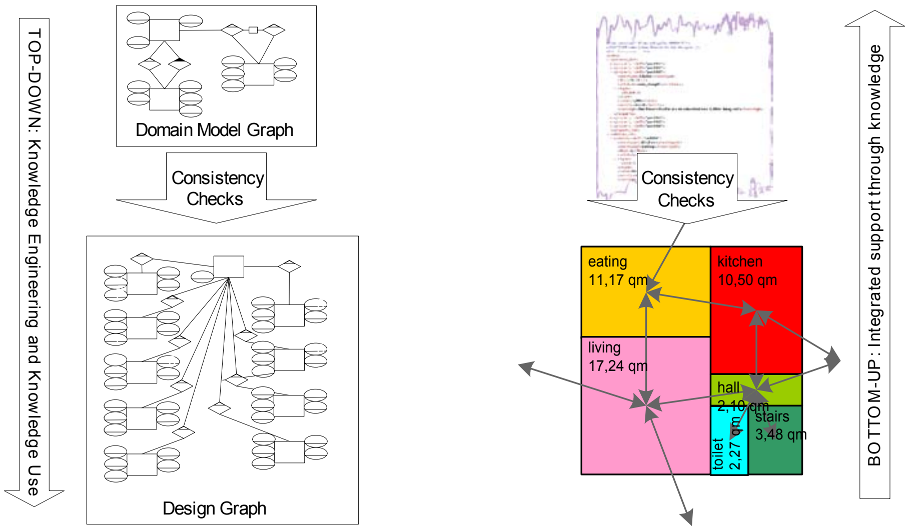

Figure 2: Complete scenario with top down and bottom up approach

In the long run it is planned to integrate both approaches. We hope to achieve the generation of the conceptual extensions of ArchiCAD from the conceptual knowledge elaborated by a knowledge engineer using the top-down tools [KRAFT2002A].

Graph based tools for conceptual design

We use the graph rewriting system PROGRES [PROGRES1999] to develop graph based tools. PROGRES allows to formally specifying graph schema consisting of attributed, typed nodes and edges. Transactions allow executing graph transformations and tests, and to dynamically create and modify a graph structure. A code generator creates C-code out of the PROGRES specification. The UPGRADE framework [JÄGER2000] provides a universally and extensible platform for graph based application, generated code is integrated and executed in so called *prototypes*. Starting from an initial prototype, extensions can be introduced to adapt the representation to the need of an application domain. Most of the projects in our group use graph technology based on PROGRES and UPGRADE to model and solve problems from different application domains [NAGL1990]. In AHEAD [SCHLEICHER2002] we develop a management system for the development processes. Integration tools [BECKER2001] are developed to keep documents from different applications consistent; we develop graph based tools to support the understanding and restructuring of complex legacy telecommunication systems [CREMER1999; MARBURGER2003] or for author support, while writing technical documents [GATZEMEIER2000]. All these projects use PROGRES to store the specific knowledge of the application domain as the schema of the graph, comfortable UPGRADE prototypes allow an abstract and problem-oriented view on the graph structure.
In the top-down approach, drawn on the left side of Figure 2, we also use graph technology to build up graph based tools for the conceptual design phase. A graphed

based editor for the *knowledge engineer* provides a possibility to define *domain specific knowledge*, here rules and restrictions about a specific building type. As the usual architect is not familiar with graph technology, the prototype's user interface presents a filtered view on the graph and methods to create and change the graph structure. A second graph-based editor enables the *architect* to define the *conceptual design* of a building project.
The graphs we use graphs to support the conceptual design correspond to the high abstraction level of this phase. Important entities for the conceptual design are the complete building, stories and rooms. Material definitions, wall constructions or exact dimension are not yet defined and need not to be considered. Relations between entities describe functional connections. Using graph nodes as rooms and edges as relations, a graph depicts a concrete building organisation. A graph that represents an actual sketch of a building is called *design graph*, depicted in the lower left corner of Figure 2.
An additional application of graph technology in conceptual design is knowledge representation. In this context knowledge consists of technical restrictions, e.g. minimal and maximal dimensions, legal restrictions, e.g. escape routes, functional requirements, e.g. the flow of traffic, economical and design restrictions. Moreover the wishes of the investor and architects personal favours have to be observed. These restrictions are represented by another graph that does not describe a concrete building, but universally applicable regulations and restrictions about a special building type. Graph nodes now represent room types, edges describe relations between room types; this graph is called *domain model graph*.
Using both graphs together allows architects to specify rules in a formal way and to use these rules during the design of a building. The knowledge stored in the domain model graph can be used for all buildings of the described building type; incremental checks guarantee the consistency between the defined knowledge and the actual sketch.

Modelling Conceptual Knowledge

The domain model graph stores structural knowledge about a building type, inserted by a knowledge engineer. To generalize from the concept of rooms, we introduce *areas*. Usually an area is a room, it may, however, be a part of a room (a big office may be composed of personal office areas) or a grouping of rooms (a chief officer area may contain a secretariat, a personal office for the chief officer, and a meeting room)
Area models describe knowledge on the *type level* to define universally valid rules for all areas of the same type. *Attributes* allow defining concrete aspects of an area, e.g. required or forbidden equipment. As we describe knowledge about a building type and not a sketch, the definition is based on area models. An attribute linked to an area model prescribe, that in a future building, all areas of this area model have to consider the attribute's content. Relations describe functional connections between area models. While *obligatory relations* demand all areas of the linked area models to have a relation installed, a *forbidden relation* does not allow any area to have such a relation installed. The definition of area models, relations and attributes is the first

step in the knowledge definition process; these basic elements are then used to create the domain model graph.
Figure 3 shows a screenshot of the domain model graph editor, the workspace of the knowledge engineer. The screenshot shows an example domain model graph on the right side and two trees containing relations and attributes. In this example attributes electricity, network, sanitary, phone, and dimensions restrictions are already defined just as the obligatory relations access and view. The graph depicted on the right side of the screenshot defines a domain model graph for a small office block, five area models are defined. The secretariat model in this example has attributes to demand a number of phone, network and electricity sockets, moreover dimension restrictions are defined. The 2 person office model has the same attributes as the need of these room types are the similar; however, the dimension restrictions will be different. Sanitary installation is required in toilets, in the graph again defined by an attribute.
The secretariat model is linked with the corridor model and the chief office model. In the building type we restrict in this graph, chief offices do not need to have an access to the corridor, but they must have access and view to the secretariat. The secretariats must have access to the corridor, just as 2 person offices and toilets.

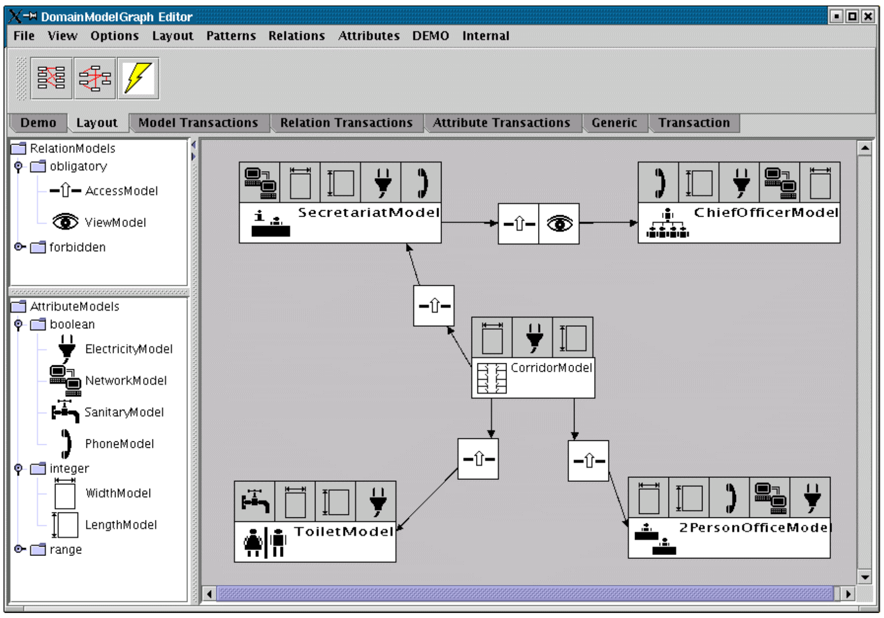

Figure 3: Domain Model Graph Editor to insert knowledge

Once defined, the domain model graph serves as a knowledge base for all building projects of this building type. Functional organisations, traffic flow, equipment, technical and legal restrictions of a small office building are then stored in the graph, the visualization provides a clear presentation and enables browsing though the knowledge database.

Using the knowledge during design

The design graph allows to evaluate the specified knowledge and to test if the defined rules are applicable to a building design. Further, and this is the main application, the design graph allows recording the structure and requirements of a building in a very early design phase. Requirements consist of investor's wishes e.g. the number of offices and size of usable area, dimensions and equipment of rooms, and relations between rooms. The specification of a building without considering any layout and material aspects allows the investor to define the usage of a building on a high abstraction level. During the subsequent constructive design, this definition can be matched with an actual floor plan to discover misunderstandings and design errors.

The design graph editor is the workspace of an architect; it supports him during the conceptual design of a building. If there exists a domain model graph as a knowledge base for a specific building type, the architect profits directly from the specified knowledge. The area models, relations and attributes defined by the knowledge engineer serve now as input for the design graph editor. The underlying building type of the domain model graph shown in Figure 3 is an office block, when constructing a building of this type, the design graph editor imports the building type specification of area models, attributes and relations. Using these predefined basic elements, in represented by the trees on the left side of the screenshot, the architect sketches an actual building by selecting an element from the tree and inserting it into the sketch. In contrast to the domain model graph, the design graph represents a building; each graph node represents an area, the number of graph nodes corresponds to the number of rooms in the future building.

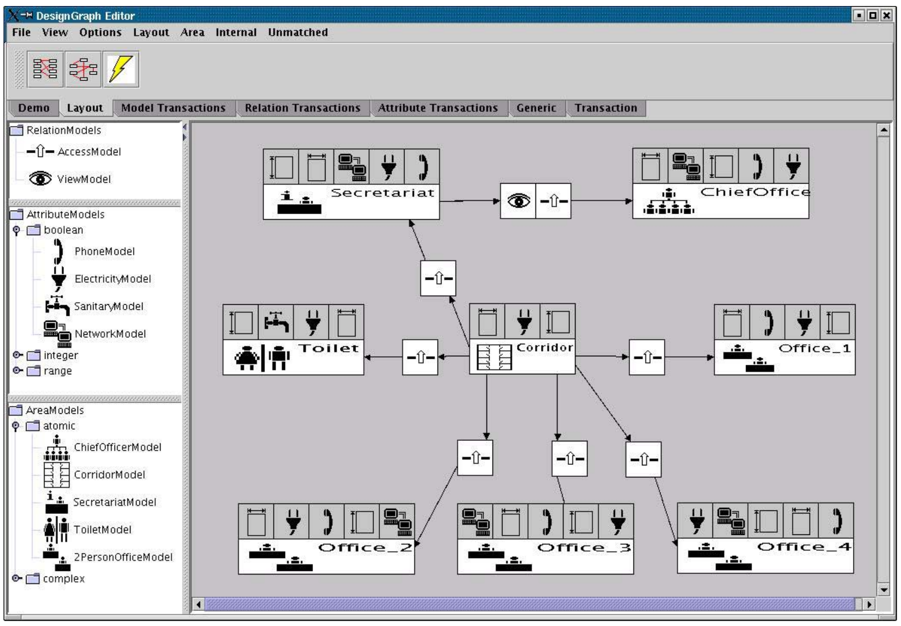

Figure 4: Design Graph Editor to use knowledge during construction of a building

The design graph, depicted on the right side of represents a part of an office building, sketched by an architect. The corridor in the middle of the graph has direct access to all areas, except to the chief office, which is only accessible from the secretariat. An access relation is symbolized by an arrow icon between two areas. Attributes define the existence of equipment in this area (e. g. the chief office has electricity, network access and a phone installed, furthermore length and width defined). As an office block consists of several offices, the 2 person office model is multiply used, each office is again described by attributes so that equipment and dimensions of the offices can differ.

Consistency Analyses between knowledge and design

The top down approach described in this paper consists of two parts, the domain model graph and the design graph. Whereas the domain model graph is used to store knowledge about building types, the design graph provides a data structure to sketch an actual building. To be able to profit from the specified knowledge, consistency analyses inform the architect if rules that are defined in the domain model graph are violated.

The design graph is in an inconsistent state, if areas, attributes or relations are used in the design graph without being defined in the domain model graph; we call these errors a *structural inconsistency*. While designing a building an architect should be able to use areas even if they are not yet defined; they can be defined later or keep undefined if no knowledge support is wished. Therefore, we tolerate structural inconsistencies as we do not want to block the architect's creativity. However, they are found and marked with an error message.

The second category of inconsistencies is based on violations of the defined rules. The attributes of an area model prescribe the usage of an area in the design graph. A secretary office should have a specified number of network sockets. If the attribute network socket is missing or the number of planned network sockets remains under the defined boundary, the design graph contains inconsistencies. Because inconsistencies are based on the content of the domain model graph, they are called *regulation violation*.

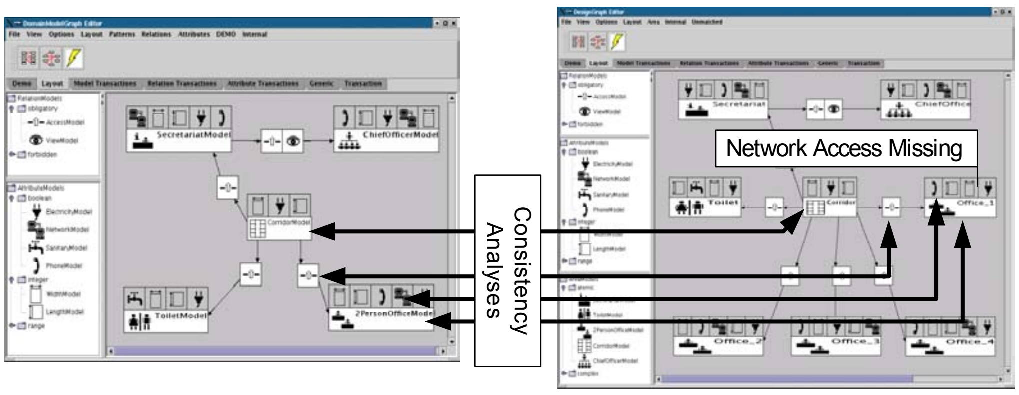

Figure 5: Consistency Analyses between Domain Model Graph and Design Graph

An example of a consistency analysis is depicted in Figure 5. As defined in the model domain graph each 2 person office should have the above defined attributes and an *access relation* to the corridor. To check the consistency, each 2 person office in the design graph is examined, if it has (a) access to the corridor, (b) all the demanded attributes, and (c) none of the forbidden ones. As shown in Figure 5, one

of the 2 person offices has no network access; this inconsistency is found and marked by an error message.
The architect has now several possibilities to react on the error message. He can fix the error by adding the missing attribute; the design graph is then again consistent with the domain model graph, and the error message disappears. He can further ignore the error message and stay in an inconsistent state, or he can confirm the sketch as correct and define exceptions from the rules.

Conceptual design support in CAD systems

The bottom-up approach is more industrially oriented; our goal is here to provide a high level support for architects during the sketch in commercial CAD-systems. We extend the CAD-system ArchiCAD [GRAPHISOFT2002] with additional features to enable the architect to conceptual design, integrated in a tool he is familiar with. Integrated in ArchiCAD, the effort to learn handling new features is minimized; further, integration provides the advantage that constructive design and visualization are already realized.

Semantic objects in ArchiCAD

In the first step we developed a *room object* to allow the architect to abstract from the wall structure to the semantic higher level concept of rooms. The room object is represented by a rectangle; a room type description and dimension information are depicted inside it. Each room type e.g. corridor or secretariat is represented by an instance of a room object, they are distinguished by fill pattern, background colour, and default dimensions. Even if the room object is currently restricted to rectangular forms, designing with typed objects is more flexible than classic designing with a wall structure. Moving or resizing a room object is much easier than changing the corresponding wall structure. The future usage of a building can be considered during the design process. Our room objects are developed with GDL [GRAPHISOFT1999], a BASIC-like language provided by GRAPHSOFT that is usually used to create furniture objects or sanitary installation. As the architect is familiar with GDL objects, the handling of room objects is very intuitively, further, a 3D representation allows estimating proportions. Actually we have a set of room objects for a single family house and a small office building.
In the second step we introduced a possibility to define relations between room objects, we call them *room links*. As we abstract form constructive elements, relations do not describe the realization but the *semantic concepts* defined between two room objects. An access relation, defined between two rooms is usually realized by a door, it can, however, be realized by a hole in the wall or by leaving out the complete wall.
In ArchiCAD, relations are depicted as labelled arrows connecting the centres of two room objects; if the architect drags a room object, the room link is automatically redrawn. The technical realization of room links is done with the aid of an application programming interfaced called C-API [GRAPHISOFT1999] provided by GRAPHISOFT and again GDL.
Using room objects and room links, the architect can sketch a building and define the functional relations of it. In fact, the sketch with room objects and room links

describes a graph, where room objects represent graph nodes and room links represent edges. Without being aware of it, the architect builds up a graph, he uses ArchiCAD as graph editor. The result corresponds to the above described design graph, in addition room arrangements and dimensions can be fixed.

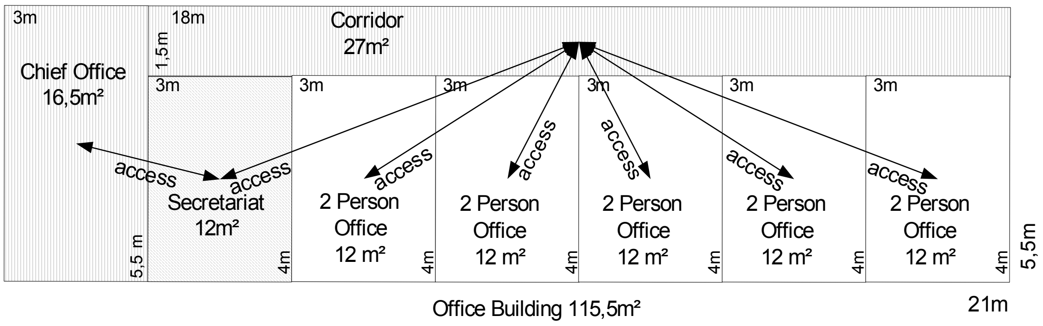

Figure 6: Conceptual Design in ArchiCAD

In Figure 6 the sketch of a small, one floor office building with room objects and room links is depicted. We see again the room types corridor, chief office, secretariat and 2 person office, just as in Figure 4. Each room object shows information about its size, horizontal and vertical dimension, furthermore information about the complete building size displayed and automatically calculated. We also see room links in Figure 6, they define here accessibility between the rooms and the corridor. As defined in the domain model graph (c.f. Figure 3) each 2 person office and secretariat should be accessible from the corridor, the chief office should be accessible from the secretariat.

Integration between conceptual and constructive design

To bridge the gap between conceptual design and traditional constructive design a new tool called *wall generator* automatically transforms the sketch with room objects and room links into an initial ArchiCAD wall structure. Wall dimensions are defined before sketching, so that dimensions depicted in the room objects are real space in the future building. Currently, we just distinguish between inside and outside walls, load-bearing and partitions are not yet considered. Starting from the generated wall structure, the architect defines materials of the walls, of course his also free to change a wall thickness or to move a wall.

Constraint checker to run consistency analyses in ArchiCAD

The intuitively way of editing with room objects is not the only advantage of the conceptual design extensions in ArchiCAD. As the architect designs using objects of a defined type, consistency analyses can be incrementally executed to examine the sketch and notify violations of the defined rules. Currently, rules are defined in a XML file manually edited; in the long run we will generate the rule definition from the domain model graph, described in the top-down approach.

The constraint checker examines at runtime each room object and room links between them. In the rule base, rules are defined about a single room object, e.g. length and with restrictions and minimal area. In contrast to the design graph, room

objects are now drawn with a defined size, so that this additional information can be now analysed. The existence of obligatory room links can be analysed, and, using the geometric information, the correctness of the drawing. E.g. when an access relation is defined by a room link, the corresponding room objects have to be neighbouring.
If the constraint checker discovers an error, a *notification object* is displayed, related to the position where the error occurred and containing an error message and a reference. The architect has again the possibility to fix the error, to ignore the error message or to confirm his sketch as correct so that the error message will disappear. The technical realization of notification objects is again done with GDL; the constraint checker is based on the C-API.

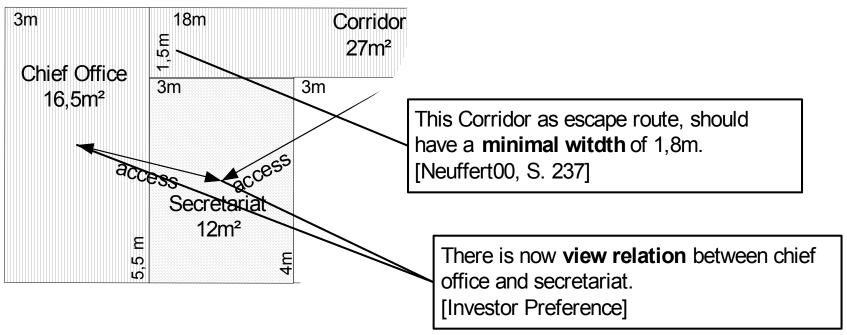
3m
18m
Corridor
27m²
1,5m
Chief Office
16,5m²
3m
3m
access
access
Secretariat
12m²
5,5 m
4m
This Corridor as escape route, should have a **minimal witdth** of 1,8m. [Neuffert00, S. 237]
There is now **view relation** between chief office and secretariat. [Investor Preference]

Figure 7: Consistency Analysis of the sketch in ArchiCAD

In Figure 7 we see a cutting of the above introduced office building sketch after the constraint checker has been run. As rule base we assume here two sources, the domain model graph depicted in Figure 3, and [NEUFERT2000] a German reference book for technical restrictions. Two inconsistencies have been found in the sketch. In a typical office building, the corridor is the main escape route; legal restrictions define its maximal length and minimal width. The corridor in Figure 7 does not fulfil the recommendation that an escape route should have a minimal width of 1,8m. The architect is pointed to that design error by an error message and a reference. The second error is based on the investors wishes defined in the domain model graph. The graph depicted in Figure 3, demanded a view relation between the chief office and the secretariat, in our example sketch only an access room link has been installed. Again, the error found by the constraint checker and visualized by a notification object.

Summary

In this paper we discussed graph based tools to support architects during the conceptual design phase. Conceptual Design is defined before constructive design; the used concepts are more abstract. We develop two graph based approaches, a top-down using the graph rewriting system PROGRES and a more industrially oriented approach, where we extend the CAD system ArchiCAD. In both approaches, knowledge can be defined by a knowledge engineer, in the top-down approach in the domain model graph, in the bottom-up approach in the in an XML file. The defined knowledge is used to incrementally check the sketch and to inform the architect about violations of the defined knowledge. Our goal is to discover design error as soon as possible and to support the architect to design buildings with consideration of conceptual knowledge.

Related Work

There are several approaches to support architects in design. Christopher Alexander describes a way to define architectural design pattern [ALEXANDER1995]. Although design pattern are extensively used in computer sciences, in architectural design this approach has never been formalized, implemented and used. In [GIPS1972] Shape Grammars are introduced to support architectural design, e.g. the design of Queen Ann Houses. The concept of shape grammars is related to graph grammars. However this approach rather supports a generation of building designs than an interactive support while designing, what we propose. Graph rewriting has been used by [GÖTTLER1990], to build a CAD system that supports the design process of a kitchen. In contrast to our approach, the knowledge is hard-wired in the specification. In [BORKOWSKI2002; GRABSKA2002] graph grammars are used to find optimal positions of rooms and to generate an initial floor plan as a suggestion for the architect. Formal concept analysis [STUMME2000] and conceptual graphs [SOWA1984] describe a way to store knowledge in a formally defined but human readable form. The TOSCANA system describes a system to store building rules; this approach is related to our research, but is based on another graph formalism.

References

ALEXANDER1995	Alexander, C.: *Eine Mustersprache.* Löcker, (1995)
BECKER2001	Becker, S., Jäger, D., Schleicher, A., and Westfechtel, B.: *A Delegation Based Model for Distributed Software Process Management.* In: Ambriola, V.: Proc. of the 8th. Europ. Workshop on Software Process Technology (EWSPT), LNCS 2077, pages 130-144, Springer, Berlin, Heidelberg, New York (2001)
BORKOWSKI2002	Borkowski, A., Schürr, A., and Szuba, J.: *GraCAD - Graph-Based Tool for Conceptual Design.* In: 1st International Conference on Graph Transformation Barcelona, to appear (2002)
CREMER1999	Cremer, K.: *Anwendung von Graphentechnik zum Reverse Engineering und Reengineering*, Diss., RWTH Aachen, Deutscher Universitätsverlag, Wiesbaden (1999)
GATZEMEIER2000	Gatzemeier, F.: *Patterns, Schemata, and Types - Author Support Through Formalized Experience.* In: Int. Conf. on Conceptual Structures (ICCS2000), LNAI 1867, pages 27-40, Springer, Heidelberg (2000)
GIPS1972	Gips, J. and Stiny, G.: *Shape Grammars and the Generative Specification of Painting and Sculpture.* In: Freiman, C. V.: Proceedings of IFIP Congress 71, pages 1460-1465 (1972)

GÖTTLER1990	Göttler, H., Günther, J., and Nieskens, G.: *Use Graph Grammers to Design CAD-Systems.* In: Rozenberg, G. and et al.: Graph Grammers and Their Application to Computer Science, LNCS 532, pages 396-409, Springer (1990)
GRABSKA2002	Grabska, E. and Palacz, W.: *Floor Layout Design with the use of Graph Rewriting System PROGRES.* In: 9th Intern. Workshop of the European Group for Intelligent Computing in Engineering, Fortschritt Berichte VDI (2002)
GRAPHISOFT1999	GRAPHISOFT: *General API Development Kit 2.2* in GRAPHISOFT, Budapest
GRAPHISOFT2002	GRAPHISOFT: *Homepage*, www.graphisoft.de (11-3-2002)
JÄGER2000	Jäger, D., Schleicher, A., and Westfechtel, B.: *UPGRADE: A Framework for Building Graph-Based Software Engineering Tools*, Technical Report AIB 00-3, RWTH Aachen, Aachen (2000)
KRAFT2002A	Kraft, B., Meyer, O., and Nagl, M.: *Graph Technology Support For Conceptual Design In Civil Engineering (invited lecture).* In: 9th Intern. Workshop of the European Group for Intelligent Computing in Engineering, Fortschritt Berichte VDI, Darmstadt (2002)
MARBURGER2003	Marburger, A. and Westfechtel, B.: *Tools for Understanding the Behavior of Telecommunication Systems.* In: Proc. of the 25th. Int. Conf. on Software Engineering (ISCE'2003), pages 430-441, IEEE Computer Society Press, Portland (2003)
NAGL1990	Nagl, M.: *Characterization of the IPSEN-Project.* In: Madhavji, N. and et al.: Proceedings of the 1st International Conference on Systems Development Environments & Factories, pages 141-150, Pitman Publishing, London, UK (1990)
NEUFERT2000	Neufert, E.: *Bauentwurfslehre.* Vieweg, Wiesbaden (2000)
PROGRES1999	PROGRES Group: *The PROGRES Language Manual 9.x* inLehrstuhl für Informatik III, RWTH Aachen (1999)
SCHLEICHER2002	Schleicher, A.: *Roundtrip Process Evolution Support in a Wide Spectrum Process Management System*, Diss., RWTH Aachen, Dt. Universitätsverlag, Wiesbaden (2002)
SOWA1984	Sowa, J.: *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley, (1984)
STUMME2000	Stumme, G., and Wille, R.: *Begriffliche Wissenverarbeitung.* Springer, Heidelberg (2000)