AGTIVE-2003 - Application of Graph Transformations with Industrial Relevance
LNCS 3072, eds. J. Pfalz, M. Nagl, und B. Böhlen, Seite 434-439, Springer

Conceptual Design Tools for Civil Engineering

Bodo Kraft

Department of Computer Science III, RWTH Aachen University
Ahornstrasse 55, 52074 Aachen, Germany
kraft@i3.informatik.rwth-aachen.de

Abstract. This paper gives a brief overview of the tools we have developed to support conceptual design in civil engineering. Based on the UPGRADE framework, two applications, one for the knowledge engineer and another for architects allow to store domain specific knowledge and to use this knowledge during conceptual design. Consistency analyses check the design against the defined knowledge and inform the architect if rules are violated.

1 Introduction

Conceptual design in civil engineering is a vague and creative phase at the beginning of the construction process. Whereas in later phases only a small part of the building is the matter of elaboration, conceptual design considers the *whole building*, its *usage*, and *functionality*. Classical CAD systems give the architect less support in this early design, as they do not model conceptual design information, such as *areas* or *accessibility* between them. As a consequence, architects currently still draw conceptual sketches by hand without any support by a computer program. In a second step, they manually transfer the conceptual design using a CAD system into the constructive design. The drawback of this development process is not the creative and artistic way of designing, but the informal way of information storage and the lack of consistency analyses [1].

In this paper we introduce *two graph-based tools* to support conceptual design. Knowledge about a specific type of buildings can be inserted by a *knowledge engineer*, using the so called *Domain Model Graph Editor*. This editor allows to *formally define* rules and restrictions about a specific type of buildings, in this paper an office block. Using the *Design Graph Editor, architects* profit from this knowledge while designing an abstract sketch of an *actual building*. The Design Graph Editor imports the defined knowledge and supports the architects by *consistency checks*, design errors are found in this early phase.

We use the graph based tools developed at our department to create new applications. Starting from an initial version, an UPGRADE prototype [2] can be extended with new functionality and adapted to the needs of an application domain. As a result, the PROGRES specification [3] can be executed in a problem oriented visual tool.

In this paper we concentrate on the provided functionality of the developed tools and how they give support. The underlying PROGRES specification and discussions of related work can be found in [4].

2 Tools for the knowledge engineer

Fig. 1 depicts a screenshot of the *Domain Model Graph Editor*, the tool used by the *knowledge engineer*. It is used to define domain specific knowledge, like room types, relations between rooms and room attributes. We call this knowledge *basic models*. Based on these models, the knowledge engineer defines a specific rule base, valid for one building type. The domain model graph in the screenshot represents knowledge about an office building, it can be used for each building project of this building type.

The editor is divided into two parts: On the left side two tree views represent relation and attribute definitions. The main part of the application shows the *domain model graph*, the data structure which represents specific domain knowledge. The first step of knowledge input is the definition of relations and attributes. In the upper tree view in Fig. 1, two obligatory relations, *access* to demand accessibility, and *view* to denote visibility between two areas, have already been defined. Integer attributes like *length* or *width* are used to demand size restrictions. Boolean attributes like *network* are used to demand or forbid an area to have a certain equipment installed, e. g. network sockets.

In the next step the knowledge engineer creates *areas*, in Fig. 1 e. g. the secretariat's room, the chief officer's room or the corridor. All room types that are needed for the described building have to be represented by an area. For each area, a minimal and a maximal number of occurrences in the building can be defined, to express that certain rooms have to be existent, or that others have to be unique. The definition of knowledge is refined through attributes, e. g. *network* or *electricity*, and relations. To demand

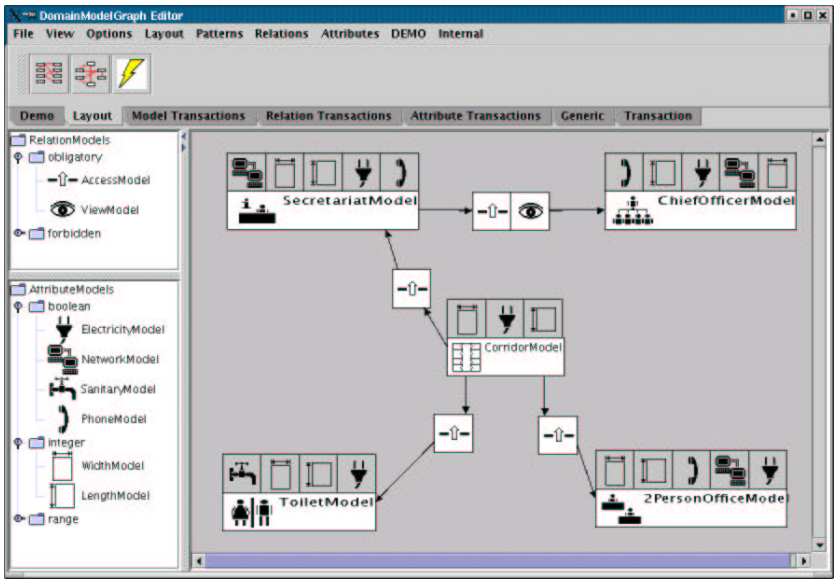

Fig. 1. Domain Model Graph Editor, a tool for the knowledge engineer

e. g. two *areas* to have an access relation, an edge-node-edge construct is established between e. g. the corridor and the secretariat's office.

The knowledge stored in the domain model graph describes rules on a *type level*, and not for an actual building. Therefore the obligatory access relation between the 2PersonOffice and the Corridor expresses that in a building each 2PersonOffice has to have an access relation to a Corridor. Forbidden relations in the domain model graph express that a relation must not be established between two areas of a building. In the same way, the attributes do not describe restrictions for an actual area, they describe restrictions valid for all areas of the specified type in an actual building.

3 Tools for the Architect

The *Design Graph Editor* is the tool used by *architects* supporting them during the conceptual design phase. This tool allows the architect to concentrate on the coarse organization and functionality of the building without being forced to think about exact dimensions or material definition. If there exists a *domain model graph* as a knowledge base for a specific type of buildings, the architect directly profits from the specified rules.

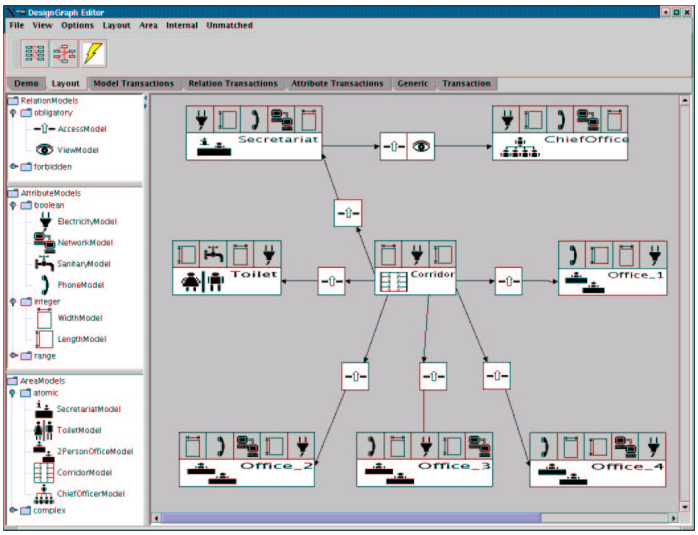

Fig. 2. Design Graph Editor, architect's tool to sketch buildings

The underlying building type of the domain model graph shown in Fig. 1 is an office building, it has been linked to the Design Graph Editor in Fig. 2. The predefined basic models are represented by the trees views on the left side of the screenshot. Using them, the architect sketches a *building* by selecting a model from the tree and inserting it into

the sketch. In our idea conceptual design is based on rooms. Thus the architect initially inserts the rooms of the future building in the design graph. He then creates attributes and relations to further define the organization of the building.

The design graph, depicted on the right side of Fig. 2 represents a part of an office building, which has already been sketched by an architect. The corridor in the middle of the graph has direct access to all areas except to the chief officer's room, which is only accessible from the secretariat's room. Attributes define the existence of equipment in corresponding area, in Fig. 2, the chief officer's room has electricity, network access and a phone to be installed, furthermore some length and width restrictions.

The layout of the nodes does not represent the arrangement of the rooms in the future building, it is optimized to provide a good readability. Even if this sketch does not contain any geometric information about the sketch, the organization of the building becomes clear.

4 Consistency Analyses

Fig. 3 depicts both tools, the *Domain Model Graph Editor* and the *Design Graph Editor* next to each other. The knowledge definition on *type level* is depicted on the left side, the conceptual design on the *level of instances* on the right side. With the aid of consistency analyses the actual design is checked against the knowledge. These checks enclose the correct usage of the defined attributes and relations, as well further as internal consistency checks.

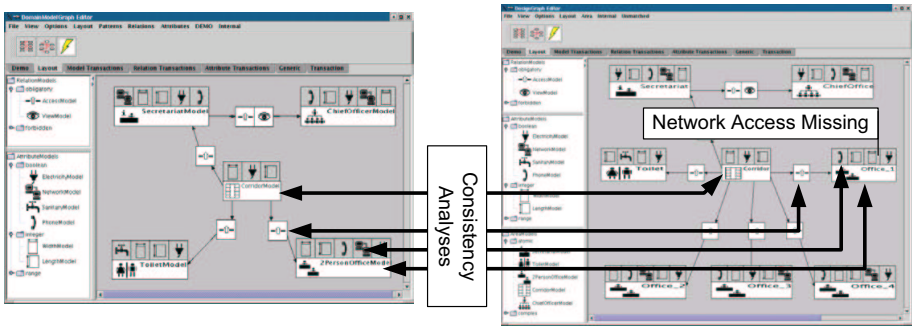

Fig. 3. Consistency checks between the Domain Model Graph and the Design Graph

An example of a consistency analysis is depicted in Fig. 3. As defined in the domain model graph each 2PersonOffice should have telephone, network, and electricity attributes, and an access relation to the Corridor. To check the consistency, each 2PersonOffice 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 Fig. 3, one of the 2PersonOffices does not have a network access. This inconsistency is found and

marked by an error message. The architect can react to this inconsistency in different ways. He is free to stay in an inconsistent state, he can fix the error in the design graph, or he can change the knowledge definition in the domain model graph.

The consistency checks are realized by graph tests comparing the domain model graph against the design graph. Notifications are created by transactions and productions, a layout algorithm ensures that they are drawn next to the inconsistent sub graph. The implementation of the consistency analyses is described and illustrated on an example production in [4].

5 Specification and Tool Construction

Both tools are the result of a tool construction process using PROGRES and UPGRADE. Based on a parameterized specification [4], the PROGRES system generates C-code. This code is then compiled, together with the UPGRADE framework, to an UPGRADE prototype. In this initial version, the prototype already provides the functionality to execute productions and transactions, and to visualize the graph. Moreover, some basic layout algorithms, filter definitions, and node representations can be used. As UPGRADE prototypes are planned to become extended, further adoptions can easily be done [5]. The tools depicted in Fig. 1 and Fig. 2, are both extended UPGRADE prototypes. Unparsers written in Java change the representation of sub graphs e. g. into a table or a tree structure. New layout algorithms ensure that attributes are arranged next to the corresponding area, and that the areas themselves are clearly arranged. New node representations provide e. g. that an icon illustrating the attribute is drawn. A further extension is an import/export interface with an automatically generated HTML based documentation. Our goal is to give the user of our tools an abstract and more clear view on graph structure, adopted to the need of the application domain. Even if the screenshots do not look like graph editors, the underlying data structure is still a graph.

In a usual PROGRES specification, the domain knowledge, here area types, attribute types and relation types are fixed in the schema and in the operational part of the specification. Node types for access or view, for each relation type, and for each room type would exist. The disadvantage of the traditional specification method is the difficulty to extend or change the knowledge. If the knowledge engineer wants to use a new room type, e. g. *Tea Kitchen*, the specification has to be changed, new code has to be generated, and the UPGRADE prototype has be restarted. We want the knowledge engineer to elaborate the knowledge, e. g. which area types are necessary. Using the traditional specification method, the tool construction would have to be repeated several times. Moreover, a graph technology expert would have to assist the knowledge engineer, helping him to change the specification and to rebuild the tools. Therefore we introduced a new parameterized specification method [4] which allows storing the specific domain knowledge in the host graph.

We use the described prototypes as an experimentation platform, to elaborate adequate data structures and functionality for conceptual design support. In the long run our goal is to extend commercial CAD systems with functionality to semantically check the architect's sketch against formally defined knowledge.

References

1. Kraft, B., Meyer, O., Nagl, M.: Graph technology support for conceptual design in civil engineering. In Schnellenbach-Held, M., Denk, H., eds.: Proceedings of the 9^{th} International EG-ICE Workshop, VDI Düsseldorf, Germany (2002) 1–35
2. Jäger, D.: UPGRADE - A framework for graph-based visual applications. In Nagl, M., Schürr, A., Münch, M., eds.: Proceedings Workshop on Applications of Graph Transformation with Industrial Relevance. Volume 1779 of LNCS., Kerkrade, The Netherlands, Springer, Berlin (2000) 427–432
3. Schürr, A., Winter, A.J., Zündorf, A.: PROGRES: Language and Environment. In Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook on Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools. Volume 2. World Scientific, Singapore (1999) 487–550
4. Kraft, B., Nagl, M.: Parameterized specification of conceptual design tools in civil engineering, this volume (2003)
5. Haase, T., Meyer, O., Böhlen, B., Gatzemeier, F.: A domain specific architecture tool: Rapid prototyping with graph grammars, this volume (2003)