UML-based Modeling of Architectural Knowledge and Design

M. Kirchhof, B. Kraft
Department of Computer Science III, Aachen University of Technology
Ahornstrasse 55, 52074 Aachen, Germany
(kirchhof|kraft)@i3.informatik.rwth-aachen.de

Abstract

We introduce a UML-based model for conceptual design support in civil engineering. Therefore, we identify required extensions to standard UML. Class diagrams are used for elaborating building type-specific knowledge: Object diagrams, implicitly contained in the architect's sketch, are validated against the defined knowledge. To enable the use of industrial, domain-specific tools, we provide an integrated conceptual design extension.

The developed tool support is based on graph rewriting. With our approach architects are enabled to deal with semantic objects during early design phase, assisted by incremental consistency checks.

1 Introduction

When designing a building, an experienced architect implicitly applies his aggregated knowledge to the new sketch. 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. 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.

Existing CAD systems give no support for this conceptual design, which is a creative process. There is no smooth transition to constructive design. The architect manually elaborates the constructive design, now using a CAD system. He manually replaces the functional elements of the sketch by constructive ones, e.g. ventilation by a window, the access by a door etc, without explicitly storing information about this transformation process. 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 phase. The modified conceptual data are lost again after the next transformation step. Such iterations are risky and expensive in terms of time and money. So, we want to enable the architect to work with semantic objects instead of syntactic elements.

The Unified Modeling Language (UML) [1] is mainly used for the requirements engineering and the design of complex software systems. Different diagram types reflect the specific aspects of different development phases and modeling tasks. We use UML class diagrams and object diagrams in a new application domain, namely the *conceptual design of buildings*. The above mentioned restrictions can be described with the aid of UML. We introduce a *graph model for semantic objects*. Based on our graph model, a *knowledge engineer* is able to set up a class hierarchy consisting of semantic objects, attributes and relationships between the objects. We define this as a *UML meta model* for civil engineering of a building type. One can imagine the knowledge engineer as an experienced architect who is able to formalize conceptually relevant knowledge about a building type. The knowledge is specific for a *class* of buildings, due to the significant differences between certain building types (e.g. single family houses, office buildings, sky scrapers). While the UML meta model has to obey certain constraints, we introduce tool support for this sophisticated task.

We realize a graph-based demonstrator by which a knowledge engineer can (a) develop the meta model and (b) specify *knowledge*, supported by the use of interactive visual tools. For the realization of these tools, we use the enhanced machinery developed at our department.

We further introduce new extensions to the industrial CAD tool ArchiCAD to provide means for the conceptual design. The above introduced semantic entities enable the architect to conceptually design a

building inside the CAD tool he is familiar with. The UML-based specified constraints are checked automatically against the ArchiCAD sketch and design errors are identified as soon as they occur. Thus, the architect can fix the errors or think about design alternatives in a very early design phase. A propagation of errors to later phases is avoided.

At Department of Computer Science III at Aachen University of Technology, tools for supporting development processes have been built for software engineering, mechanical engineering, chemical engineering, process control, telecommunication systems, and authoring support. In all these tools we use a graph-based tool construction procedure: internal data structures of tools are modeled as graphs, changes due to command invocations are specified by graph rewriting systems. We derive tools automatically from specifications, using the PROGRES system [3] for specification development, a code generator for producing code out of the specification, and the UPGRADE visual framework environment [4] into which the code is embedded. The resulting tools are efficient demonstrators for proof of concept purposes.

This paper is structured as follows: we first introduce an application of UML for conceptual design of buildings [5, 6]. We then derive a graph model and a UML meta model and discuss their expressive power. We will briefly describe our tool construction process and present the realized tools. We further explain how the consistency between the specified knowledge and the architect's sketch is implemented. We close with conclusion and discussion of related work.

2 UML for Conceptual Design

UML class diagrams allow structuring of information and data. Basic relationships as inheritance and association can be modeled. UML supports modeling on an adequate abstraction level.

As important semantic entities for conceptual design we identify *areas*, *rooms* and *sections*. The most important element in building design is obviously a room. We define a *room* as a space with a certain functionality, which is usually surrounded by a wall structure. Rooms are fundamental, because they represent the cognitive units of the architect and serve as basic building blocks. The set of rooms gives an almost complete picture of a building. We define a *section* as a part of the room representing a certain functionality. Using sections, a more detailed view on the usage of a room can be provided. Finally, an *area* is defined as a heterogeneous collection of rooms. In

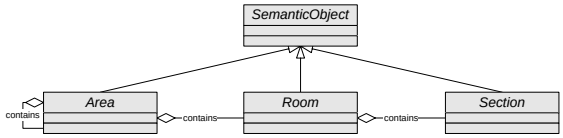

Figure 1: Graph Schema

a complete example, each floor consists of a set of different rooms, like offices, corridors and a library; one part of the library is restricted to the library's staff, the other part is public. So, the floor will be modeled as an area, the library as a room with a private and a public section.

Using UML class diagrams, we define our graph schema as depicted in Fig. 1. The super class `SemanticObject` is inherited by the the classes `Area`, `Room` and `Section`. There exists a reflexively aggregation relation of the class `Area`, because areas can be composed from several areas. Furthermore, an area can consist of several rooms, thus the class `Area` aggregates the class `Room`. Analogously, the class `Room` aggregates the class `Section`.

Based on the graph schema, a building type-specific UML meta model is represented in the class hierarchy depicted in Fig. 2. The class `Room`, defined in the graph schema, serves as a superclass for different categories of rooms. In the example, we distinguish classes for traffic rooms, work rooms, and sanitary rooms as they represent different basic functionalities. In our case, these classes are modeled as abstract classes as they should not directly be used for design. While the abstract classes serve as structurizing elements that summarize the common characteristics of a room, the non-abstract classes reflect concrete entities of a building type. Further functionalities are expressed by a hierarchy of room classes.

Up to here, we have a functional decomposition for civil engineering tasks. We use this building type-specific decomposition to describe *knowledge* in a UML class diagram. The knowledge reflects legal, economical, and technical restrictions, specified on the type level with the aid of class attributes, cardinalities and associations. The defined knowledge can later be checked against a specific building sketch.

We add certain notations for restrictions and requirements to standard class diagrams. As each architect's sketch implicitly integrates the defined entities of Fig. 2, the building sketch can also be seen as a UML object diagram. In contrast to UML used in software engineering, no strict reference between UML class diagrams and the architect's sketch (the UML object diagram) is demanded here. We rather allow the construction of arbitrary sketches, as we do not want

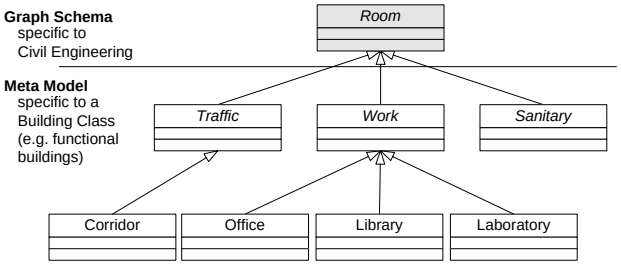

Figure 2: UML Meta Model for Conceptual Design

to restrict the architect's creativity. In a later step, not until the architect wishes, the sketch is validated against the knowledge defined in the class diagram. Inconsistencies are not automatically resolved, but informative error messages and hints are propagated into the architect's tool. He can then decide to fix or to ignore the discovered inconsistencies. Our tool support is incremental and integrated into the CAD tool, thus it allows a comfortable efficient work flow [2].

In the following, we describe the use of UML class attributes, cardinalities, and associations for knowledge engineering. In some cases, the semantics have to be extended.

Local requirements and restrictions of areas, rooms and sections can be viewed as *attributes*. In our approach they are modeled as UML class attributes. We distinguish between Boolean attributes and integer range attributes. In contrast to the semantics of standard UML class attributes, we define the semantics of attributes as follows: the attribute value in the class diagram represents a guideline, how the corresponding attributes in the concrete sketch (the UML object diagram) have to be set. Boolean attributes can demand properties to be existent, e.g. all sanitary rooms have to be equipped with sanitary installations. This fact is modeled as a Boolean attribute with the value `true`. To prohibit properties to be existent, the Boolean attribute is set to `false`. E.g., the offices should not be equipped with sanitary installations. Integer attributes are used to specify number restrictions or size restrictions. They can be associated with equality relationships, less-than relationships, or greater-than relationships. For example, each room type has a defined minimal length, width and size, which is denoted by a greater-than relationship of the respective attribute. Again, the architect is able to sketch buildings ignoring all these facts and tune it at a suitable point in time. Everything not modeled in the class diagram is optional, so the architect can set additional attributes at will. This stands again in contrast to standard UML, where the object structure is completely determined through the class diagram.

As a further extension of the UML class diagram, we introduce *class cardinalities*. They determine the minimal and maximal number of instances of a class in the object diagram. In civil engineering, the number of certain rooms is limited to a certain range. E.g., each single family house has to have one kitchen, one floor, one living room, one bathroom, and one ore more bedrooms. A guest toilet is optional, so the minimal cardinality is set to zero, the maximal cardinality is set to one. The minimal and maximal cardinality can be defined as a value or a as a complex expression. These complex expressions can reflect dependencies between the cardinality of objects of different classes in the sketch. In civil engineering such cardinality relationships between room types very often ensure the functioning of a building. To express the requirement for an office building that one toilet is needed for each five offices, the minimal cardinality of the class attribute is set to `1+(card(Office) div 5)`. With complex expressions, the expressive power of class cardinalities is leveraged to an appropriate level for civil engineering.

Besides the local requirements of the semantic entities, *relations* between them exist. In civil engineering, relations can express dependencies between areas, rooms, and sections, e.g. accessibility or neighborhood. We use UML associations between classes to reflect relations between our semantic objects. The association's name is set to the concrete relation description. We distinguish between symmetric relations, e.g. accessibility between rooms, and asymmetric relations e.g. waste air from a kitchen to the outside. These relations are modeled using UML by directed and undirected associations. Relations between semantic entities can be demanded or forbidden. For example, while the chief's office should have an access relation to the secretariat's office, it must not have a direct access from the corridor. To formalize these facts in the class diagram, we use association multiplicities. As in standard UML, upper and lower bounds specify minimal and maximal number of instances. A demanded relation between two semantic objects is expressed through a lower bound greater than zero. E.g., each office must have an access to one corridor and a corridor can have access to some offices, the resulting multiplicity is depicted in Fig. 3. A forbidden relation is expressed by an upper bound of zero, so no instances can be related. For another type of optional

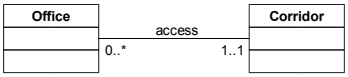

Figure 3: Association Multiplicity

relations, we redefine again the semantics of standard UML. We allow associations in the object diagram, which are not defined in the class diagram. With that extension to express optional relations, we enable the creation and reasonable usage of incomplete knowledge. This extension is required, because knowledge definition for civil engineering is a more evolutionary process than the design of software systems.

3 Tools for Conceptual Design Support

Based on the graph rewriting system PROGRES, we develop a graph-based tool for the knowledge engineer. The tool construction process works as follows: Using the PROGRES language [3], a graph schema and graph transformations are specified in a declarative way, supported by a visual editor [7]. The graph schema is fixed; based on it, graph transformations allow constructing and modifying a graph data structure at runtime. Graph tests allow querying the graph. In the specification of the civil engineering tools, the above defined graph schema (Fig. 1) is fixed. The building-specific class hierarchy, so the meta model (Fig. 2), can be elaborated at runtime. The operational part of the specification follows a new parameterized PROGRES specification method [8]. The PROGRES system provides a code generation, the generated code can be compiled together with the UPGRADE framework [9] to an external visual application called an UPGRADE prototype. This prototype allows executing graph transformations, so to create and modify a graph at runtime. Basic graph layouts are already implemented. To adapt the prototype to the needs of a special application domain, further layouts and unparsers can be integrated. The application in Fig. 4 is an UPGRADE prototype, extended to a UML-like knowledge editor, where graph nodes are layouted as UML classes.

A knowledge engineer uses this tool to define domain-specific knowledge, like room types, relations between rooms and room attributes. On the left-hand side, the tree views represent the defined semantic objects from the meta model classes, especially the rooms. Furthermore, pre-defined relation types and attribute types are depicted. The main part of the application shows the *domain knowledge graph*, the data structure which represents domain-specific knowledge. The first step of the knowledge input is the definition of the meta model, so semantic objects, relations and attributes. In the bottom tree in Fig. 4, three relations, *access* to demand accessibility, *aeration* to demand air supply, and *view* to demand visibility between two areas have already been defined. Integer attributes represent *length* or *width* restrictions. Boolean attributes like *heating* are used to demand or forbid a room to have a certain equipment installed. If a meta model has already been defined, it can be imported into the tool.

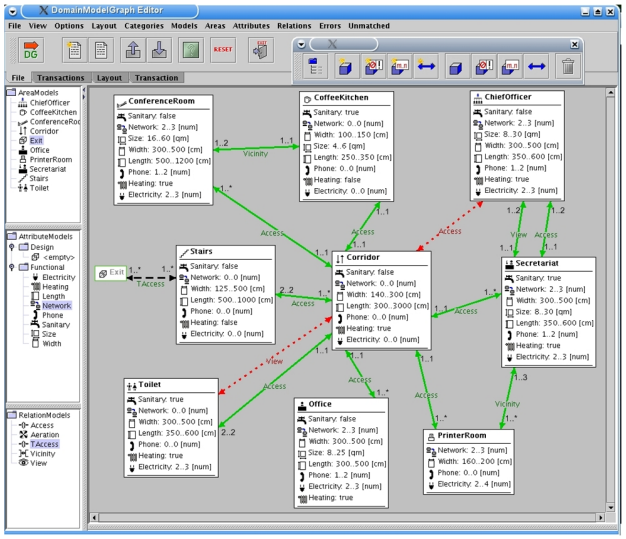

Figure 4: Domain Knowledge Graph Editor

The second step is to describe knowledge on the *type level*. The *access* relation between *Office* and *Corridor* (Fig. 4) expresses that in a building, each office has to have an access to a corridor. 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.

The so defined knowledge can be checked against a sketch in ArchiCAD. It supports architects during the conceptual design phase. Using new conceptual design objects inside of ArchiCAD, the architect can conceptually elaborate an early sketch of the building [10]. As he uses the same semantic objects as in the domain knowledge graph, namely rooms, areas and sections, he implicitly creates a UML object diagram. Room sizes and positions sketched by the architect implicitly

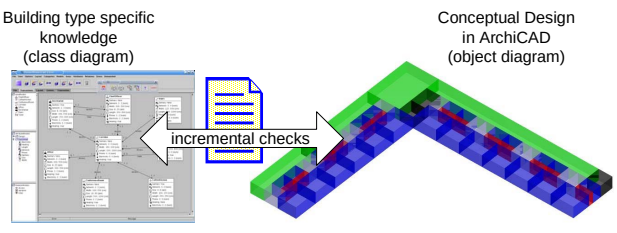

Figure 5: Tool Support integrated in ArchiCAD

determine attribute values. The consistency between the defined knowledge and the architect's sketch can now be checked.

4 Conclusion and Discussion

In this paper, we introduced the usage of UML in architectural knowledge and design. We identified semantic entities in the conceptual design of buildings and showed an adequate representation based on UML. The merits of UML are, that it is well-known by computer scientists and is heavily used in software systems design and other domains. The standardized diagram types have a good readability and many editors and tools for UML exist. The modeling task benefits from the object oriented approach, thus the inheritance relationship is one of the basic concepts. Furthermore, standardized concepts and visual representations for class attributes, inheritance of attributes, associations, aggregations, and cardinalities exist. The demerits of UML are, that the defined semantic is purely based on software systems engineering and does not perfectly cover the domain of civil engineering, especially architectural knowledge and design. Although defined semantics exist, some parts are missing: first, no meta model concept is defined. This is required to restrict classes of sketches. Second, no means for cardinalities of instances of a class are provided (e.g., for the singleton pattern). Third, the class diagram fully determines the object diagram.

The drawbacks of a UML-based approach are solved by the definition of extensions to UML. We introduced new semantics of attributes, pre-determing values if present in the architect's sketch. Boolean attributes represent specific requirements, whereas integer attributes are combined with relationship type for evaluation during consistency checks. The concept of classes is extended by *class cardinalities* with lower and upper bounds. These bounds can be atomic or complex, giving the opportunity to specify arbitrary dependencies (in terms of cardinalities) between instances of classes. For the specification of relations between semantic entities, we discussed a notation based on associations, and cardinalities of associations for demanded and forbidden features. While other constraint languages exist, we follow an approach which enables an architect, not a computer scientist, to formalize domain-specific knowledge in a visual representation. Therefore, the underlying graph technology [8] is fully hidden.

While the determination of object diagrams by class diagrams is a reasonable approach in software systems engineering, in civil engineering this viewpoint is not suitable. We define a model of a building as a set of minimal requirements for a class of buildings. Thus, the sketch contains certain elements at least, but can contain further elements, even if they are not included in the model. This applies to classes, attributes, and associations. This flexibility is required to meet the architect's need for freedom during design, as it has been shown that too tight boundaries imposed by tools led to dismissing tool support [2].

Although complex class cardinalities represent relations between two or more objects, we model them as class attributes in order to focus the number of instances to the class itself. This ensures a separation of concerns. Up to now, we applied the UML concepts of classes, attributes, inheritance, aggregation, and association to civil engineering. The semantic of UML methods is subject of future research. Ongoing research is done in the field of integrated eHome systems and eBusiness elaborating such an approach.

There are several approaches to support architects in design. Christopher Alexander describes a way to define architectural design pattern [11]. Although design patterns [12] are extensively used in computer sciences, in architectural design this approach has never been formalized, implemented and used. The object constraint language OCL [13] allows a more precise definition of UML. In our approach, the detailed formalization of constraints is done using graph transformations. The underlying constraint language is hidden in the graph specification due to usability reasons. In [14] Shape Grammars are introduced to support architectural design, e.g. the design of Queen Ann Houses [15]. 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.

Graph technology has been used by [16] 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 [17, 18], graph grammars are used to find optimal positions of rooms and to generate an initial floor plan as a suggestion for the architect. UML use case diagrams and activity diagrams are used by [19] to derive functional requirements of a building. This kind of knowledge definition is a rather implicit way and not as general as our approach. Formal concept analysis [20] and conceptual graphs [21] describe a way to store knowledge in a formally defined but human readable form. The TOSCANA system [22], based on formal concept analysis, describes a system to store building rules. In contrast to our approach, the TOSCANA system concen-

trates on collecting and structurizing statute books. It is neither optimized for the conceptual design phase nor integrated into a CAD system. Our approach aims at direct practical use for architects.

References

[1] Fowler, M.: UML Distilled. Addison-Wesley (2003)

[2] Nagl, M., ed.: Building Tightly Integrated Software Development Environments: The IPSEN Approach. LNCS 1170. Springer, Berlin (1996)

[3] Schürr, A.: Operationales Spezifizieren mit programmierten Graphersetzungssystemen. PhD thesis, RWTH Aachen. DUV (1991)

[4] Böhlen, B., Jäger, D., Schleicher, A., Westfechtel, B.: UPGRADE: A Framework for Building Graph-based Interactive Tools. In Mens, T., Schürr, A., Taentzer, G., eds.: Electronic Notes in Theoretical Computer Science, Vol. 72. Elsevier Science Publishers (2002)

[5] Kraft, B., Meyer, O., Nagl, M.: Graph Technology Support for Conceptual Design in Civil Engineering. [23] 1–35

[6] Kraft, B., Nagl, M.: Semantic Tools Support for Conceptual Design. [27] 1–12

[7] Winter, A.J.: Visuelles Programmieren mit Graphtransformationen. PhD thesis, RWTH Aachen. DUV (2000)

[8] Kraft, B., Nagl, M.: Parameterized Specification of Conceptual Design Tools in Civil Engineering. [25] 1–16

[9] Jäger, D.: UPGRADE - a Framework for Graph-based Visual Applications. [26] 427–432

[10] Kraft, B.: Konzeptueller Gebäude-Entwurf in ArchiCAD. GRAPHISOFT News **3** (2003)

[11] Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A Pattern Language. Oxford University Press, New York, NY, USA (1977)

[12] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison-Wesley (1995)

[13] Warmer, J., Kleppe, A.: The Object Constraint Language : Precise Modeling with UML. Addison-Wesley (1998)

[14] Gips, J., Stiny, G.: Shape Grammars and the Generative Specification of Painting and Sculpture. In: Proceeding of the IFIP Congress 71. (1972) 1460–1465

[15] Flemming, U.: More than the Sum of Parts: the Grammar of Queen Anne Houses, Environment and Planning B. Planning and Design (1987)

[16] Göttler, H., Günther, J., Nieskens, G.: Use of Graph Grammars to Design CAD-Systems. In: Graph Grammars and their Application to Computer Science, LNCS 532, Springer (1990) 396–409

[17] Borkowski, A., Grabska, E., Szuba, J.: On Graph-based Knowledge Representation in Design. [24]

[18] Borkowski, A., Grabska, E., Nikodem, E.: Floor Layout Design with the Use of Graph Rewriting System PROGRES. [23] 149–157

[19] Szuba, J., Ozimek, A., Schürr, A.: On Graphs in Conceptual Engineering Design. [25]

[20] Stumme, R., G. Wille, eds.: Begriffliche Wissensverarbeitung. Springer, Berlin (2000)

[21] Sowa, F.J., ed.: Conceptual Structures. Addison Wesley, Reading, MA, USA (1984)

[22] Eschenfelder, D., Stumme, R.: Ein Erkundungssystem zum Baurecht: Methoden und Entwicklung eines TOSCANA Systems. [20] 254–272

[23] Schnellenbach-Held, M., Denk, H., eds.: Advances in Intelligent Computing in Engineering. Proc. of the 9^{th} Int. EG-ICE Workshop (2002)

[24] Songer, A.D., John, C.M., eds.: Computing in Civil Engineering. Proc. of the 3^{rd} Int. Workshop on Information Technology in Civil Engineering (2002)

[25] Proc. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE 2003) to appear (2004)

[26] Nagl, M., Schürr, A., Münch, M., eds.: Proc. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE'99) LNCS 1779. Springer (2000)

[27] Flood, I., ed.: Towards a Vision for Information Technology in Civil Engineering. Proc. of the 4^{th} Joint Int. Symp. on Information Technology in Civil Engineering. CD-ROM (2003)