Advanced Engineering Informatics
eds. J.C. Kunz, I.F.C. Smith and T. Tomiyama, Elsevier, Seite 1-22, angenommen, erscheint.

Visual Knowledge Specification For Conceptual Design: Definition and Tool Support

Bodo Kraft Manfred Nagl
RWTH Aachen University

Abstract

Current CAD tools are not able to support the conceptual design phase, and none of them provides a consistency analysis for sketches produced by architects. This phase is fundamental and crucial for the whole design and construction process of a building. To give architects a better support, we developed a CAD tool for conceptual design and a knowledge specification tool. The knowledge is specific to one class of buildings and it can be reused. Based on a dynamic and domain-specific knowledge ontology, different types of design rules formalize this knowledge in a graph-based form. An expressive visual language provides a user-friendly, human readable representation. Finally, a consistency analysis tool enables conceptual designs to be checked against this formal conceptual knowledge.

In this article, we concentrate on the knowledge specification part. For that, we introduce the concepts and usage of a novel visual language and describe its semantics. To demonstrate the usability of our approach, two graph-based visual tools for knowledge specification and conceptual design are explained.

Key words: Architectural Design, Conceptual Design, Knowledge Specification, Visual Language, Graph Transformation

1. Introduction

The difficulty and complexity of *construction processes* coupled with the wider and increasing distribution and interaction of the players involved in these processes make some changes necessary. Current existing CAD tools are not able to provide adequate support for the process. There is *no tool* available supporting the *conceptual design* phase, which is fundamental and crucial. Especially, none of the currently available CAD tools enables a consistency analysis of sketches produced by architects.

To provide architects with better support in conceptual design, there is a need for novel CAD tools abstracting from constructional design. Rather than thorough and detailed plans, conceptual design focuses on the *functionality* of buildings as a whole, consisting of various functional and *interrelated entities*. A resulting and consistent conceptual sketch forms the basis of all following design stages.

This abstraction allows the identification of the organizational configuration of a building and ensures its usability. Even if the conceptual design

Email addresses:
kraft@i3.informatik.rwth-aachen.de (Bodo Kraft),
nagl@i3.informatik.rwth-aachen.de (Manfred Nagl).

phase is performed at the beginning of a construction process and the degree of details remains low, a *large number of constraints* have to be regarded, arising from various *domains*, such as legal aspects, technical, functional, and financial restrictions. The majority of these restrictions are specific to a class of buildings, e.g. office buildings, car garages, or residential buildings.

Currently, after finishing the conceptual design, architects manually transfer their sketches, usually in the form of hand drawings, into a CAD tool. The reason is that CAD tools can only handle constructive design information, e.g. used materials, constructive elements like walls and doors, and their exact dimensions. Thus, the valuable *conceptual* design *information*, i.e. all decisions about the organizational structure of the building, *gets lost*.

The *ConDes project* (Conceptual Design) at the Department of Computer Science 3 of RWTH Aachen University aims at novel concepts and support for conceptual design. The complete scenario of the project comprises *two parts*.

In the *top-down part*, we formally specify a prototype for a CAD tool, the *Design Graph Editor*, that allows to sketch conceptual designs. This tool is internally realized by graph technology [1][2][3]. A further graph-based tool, the *Knowledge Graph Editor*, allows the definition of relevant conceptual knowledge specific to a class of buildings. Knowledge that has been formalized once can be used for all building projects of the corresponding class. The effort of formalizing knowledge only pays off if the knowledge is repeatedly used. A *consistency* analysis *tool* enables conceptual designs to be checked against this defined knowledge. Notifications are given to the architect if restrictions are violated [4]. All these graph-based tools can be seen as research prototypes.

Even if the tools provide a graphical user interface and are user-friendly, there is always a big effort to learn a new tool. For this reason, in the second *bottom-up* part of the project [5], we *extend* the CAD tool *ArchiCAD* [6] to offer new functionality for conceptual design and consistency analysis. Both, the graph-based conceptual sketches as well as the graph-based conceptual knowledge can be imported in ArchiCAD due to a common XML-format. The formalized knowledge is then available in ArchiCAD, an architect can immediately start developing conceptual sketches and frequently check them.

In this article, we concentrate on the *knowledge specification part* of the project, introducing an integrated system for graph-based knowledge specification, conceptual design, and consistency analysis [4]. The main part of the article is the description of a new *visual language* for graph-based knowledge definition. We outline both, the expressive power of this visual language as well as its graph-theoretical background. To demonstrate the usability of our approach, the above mentioned graph-based tools are described and illustrated.

There are some topics not covered by this article. The graph-based specification and realization of tools is described elsewhere [7][8]. Especially, there is a need for a parameterized specification approach [7] as the underlying knowledge is changed or extended within a construction process. Also, the extension of any industrial tool the reader can find in [9]. The importance of concurrent engineering is increasing. In [10] the system architecture is therefore extended, so that different knowledge engineers can elaborate on a central knowledge base. In the extended system architecture, also different architects can access the knowledge base to check their conceptual sketches. The underlying technology is based on the CORBA platform [11].

2. Related Work

In the literature there are several approaches to support architects in design. Alexander describes architectural *design patterns* [12]. In a textual notation, he defines properties and requirements for buildings and their environment. Design patterns are extensively used in computer sciences [13]. In architectural design this approach has never been formalized, implemented, or used.

When solving a design problem, the first difficulty is to get into the new project. Joedicke identifies *three different perspectives* of a design problem [14]. First, a construction-oriented perspective focusses on the structural system of a new building. Second, the shape-oriented perspec-

tive concentrates on the creation of the outer form and the building's front. Finally, a functionality-oriented perspective regards the organization inside the building, e.g. the required rooms, their dimensions, and relationships. The functionality-oriented perspective is always specific to one class of buildings. Steinmann demonstrates that all three perspectives already exist during the conceptual design phase [15].

Most of the approaches that support architects during design use concepts from the field of *artificial intelligence* [16][17]. They can be roughly separated into evaluative and generative approaches [18].

One form of an *evaluative approach* uses a knowledge base to store information about a specific domain and design decisions made in the past together with their contexts. The system filters those parts from the knowledge base that are important for a current problem of the architect. Such systems support human decision and, therefore, are called Decision Support Systems (DSS). To know which part of the knowledge base is relevant, the design in most cases strictly follows a predefined process model (Case-based Design) [19]. In [20][21] a tool is presented that works like a CAD tool but, additionally, can identify the functional entities in a CAD sketch. Furthermore, it checks a sketch against knowledge. However, the tool is not implemented yet. The aim of [22] is to extract all relevant information, concerning legal restrictions, from a general 3D CAD model. This information should be used to check future sketches. In the ConDes project the formal and explicit representation of architectural knowledge as a basis for the consistency analysis is an essential part.

Other forms of evaluative approaches do a technical analysis of the constructive design. Whereas simulations, like daylight analysis or stability analysis, just use numerical calculations, other aspects – as the fulfillment of customer needs – again make use of knowledge bases. In [23] a new approach is presented to support the conceptual design phase in *structural engineering*. The authors demonstrate how description logic [24] can be used as a basis to formalize structural knowledge. A tool called ConEd [25] implements the approach. Analogous to the ConDes project, the aim of the authors is to formalize knowledge and to used this knowledge to support an engineer during the conceptual design phase. Opposite to the ConDes project, ConEd provides support for structural design. The developers of ConEd follow a construction-oriented perspective on the design problem, instead of a functionality-oriented perspective in the ConDes project.

Generative approaches mostly use the represented structural knowledge to create at least an initial prototypical design. Parts of that design are then further refined, by applying further generative rules. So, in these approaches it is mainly the machine that creates the design. The SEED system [26][27] provides a support for the early phase in architectural building design. Different modules – SEED-Pro, SEED-Layout and SEED-Config – allow for specifying the requirements for a building, generating floor plans, and three dimensional models based on these requirements. Knowledge specification is done in so-called specification units, storing the requirements for the future building. Although the SEED approach also provides user interaction, the generation of building sketches is the main focus. The ConDes project rather follows an interactive approach, where the designer is dominating the design process.

Most of the generative approaches are *implemented* in Prolog or Lisp and are not integrated with existing CAD tools. In [28][29][30] *graph grammars* are used to find reasonable positions of rooms and to generate an initial floor plan as a suggestion for the architect. For that, functional requirements for a building, especially the traffic flow inside, are formalized in UML use case and activity diagrams [31]. These diagrams are then mapped onto a room graph representing an abstract structure of the future building. In contrast to the ConDes approach, all knowledge is fixed in the PROGRES specification and can only be defined by a computer scientist. So, the knowledge is stored implicitly. Moreover, this approach focuses the formalization of the buildings' usage. The approach presented in this article is more flexible and allows knowledge formalization for different domains.

Stiny and Gips use *shape grammars* to formalize knowledge about different classes of buildings.

This approach again focuses on the generation of sketches [32]. Flemming defined e.g. a comprehensive shape grammar containing design rules for Queen Anne houses [33]. Although grammars are mainly used to generate designs, they can also be used to check whether a design is within a language, i.e. that a sketch can be created by a grammar. The ConDes project uses graph grammars and graph transformations to formally specify conceptual knowledge and conceptual designs. A consistency analysis based on parameterized graph transformations [8] allows for checking the knowledge against a conceptual design.

In [34] an informative overview on *knowledge representation* is given. Five distinct roles are identified and discussed to explain the term knowledge representation independently from an actual situation. A lifecycle for knowledge – definition, use, adaption, and reuse – is described in [35]. In this paper, a distinction is made between internal and external knowledge and the lack of meaningful formalization methods is identified. Finally, the CoMem system, a kind of knowledge browser, is demonstrated. Knowledge from previous projects can be stored and reused. In contrast to CoMem, the main goal of the ConDes project is the formalization of the conceptual knowledge and the possibility to automatically check a conceptual sketch against the knowledge.

The importance of *knowledge processing* for architectural design is comprehensively discussed in [19]. The fundamental techniques are described and their application in the field of architectural design is introduced. Fenves and Garrett introduce in [36] a representation model for storing *standards* based on decision tables. Their goal was, like in the ConDes project, to provide automatically checks of architectural sketches.

In [37], different *new paradigms* for a conceptual design support are proposed. Among other things, top-down decomposition, modularization, and the use of object-orientation for architectural design is introduced. Although the work is neither implemented nor integrated into a CAD tool, the ideas are fundamental for our research.

The *semantic web* approach [38] aims to improve the quality of information in the World Wide Web. This approach is based on RDF [39], a language developed especially for modeling knowledge. Formal concept analysis [40] and conceptual graphs [41], based on *semantic networks*, also describe a way to store knowledge in a formally defined but human readable form. The TOSCANA system [42], which is based on formal concept analysis, contains a tool to store legal building rules. In contrast to the ConDes project, it is restricted to store and classify texts of laws. Dependencies between rules and the internal structure of them cannot be represented. Finally, the TOSCANA system is not integrated with a CAD tool.

Nosek and Roth deliver in [43] an *empirical* survey. According to this survey, a visual representation of knowledge, using e.g. semantic nets [44], is more clearly comprehensible than a textual representation, as e.g. predicate logic. Gross and Do underline the importance of *diagrams* especially for architectural design [45].

There are some commercial implementations in the field of conceptual design support. *SketchUp* [46] is a CAD tool that allows architects to develop early sketches in a 3D view. Based on geometric shapes, as triangles, circles and rectangles, a 2D form can be designed and transformed into a corresponding 3D volume. SketchUp provides a software tool that is easy to use and that emulates the work with paper and pencil. SketchUp concentrates on the shape of the building, it focuses the *shape-oriented* conceptual design (cf. [15]). It is neither possible to represent and store the functionality of the building nor to check the consistency of the sketch against previously defined design rules.

KollabNet [47] develops a design management software to support the *distributed planning* of development processes. Different tools allow to define and execute a process model, to manage the user authorization, the version history of all existing documents, and to store the requirements of a product. Opposite to the ConDes project, KollabNet is not restricted to one application domain. The visual language of the ConDes project has been developed to enable architects and civil engineers to formalize knowledge specific to architecture. The provided elements of the visual language therefore allow to store restrictions and requirements of this application domain more precisely.

3. System Architecture

The ConDes *system architecture* describes the organization of the complete solution and the dependencies between its parts. Especially, it illustrates, how knowledge specification is embedded in the whole system. Thus, the most important components of the system are knowledge formalization and conceptual design support.

The overall goals and functionality of ConDes are the following: In conceptual design, the tools allow architects to sketch the coarse organization and functionality of a building. For that, we provide the possibility to define a set of *semantic elements* (e.g. areas, rooms, sections) specific to a class of buildings in an *ontology*. The architect can then *sketch* an actual conceptual design built-up from these ontology elements. In *knowledge formalization*, which is done by a knowledge engineer, conceptually relevant knowledge from different domains (law, economy, experience) can be formalized. Because both parts, conceptual knowledge and conceptual design, are finally available in a formal representation, the conceptual design can be *checked* and inconsistencies are presented to the architect.

Figure 1 depicts the architecture of the complete system for conceptual design support. The two *main sections* of the system are structured in two *columns*. The knowledge formalization part is on the left hand side and the conceptual design part on the right hand side of the figure.

Three horizontal layers structure the system architecture in more detail. At the top of Figure 1, general knowledge and tool functionality are depicted. They are – to a certain extend – specific for civil engineering, but they do not reflect characteristics of a class of buildings. In the middle layer the specific knowledge definition part and at the bottom the developed tool support are shown. Now, both reflect the structure of a class of buildings or make use of this structural characterization. The horizontal arrows depicted between the columns describe the necessary integration between knowledge and design on each layer. The integration allows to check the conceptual design against the knowledge. At the bottom of Figure 1, we find tool

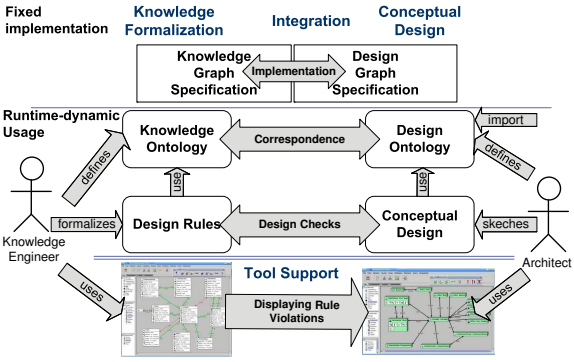

Fig. 1. System Architecture and Tool Support

functionality for making use of this knowledge in order to produce a conceptual sketch.

The first layer at the top of Figure 1 stands for a general *graph grammar specification* using the graph rewriting system PROGRES [2]. The specification comprises a graph *schema* that determines graph node classes, node attributes, and edge types to restrict the valid graph class to the needed subset [9]. Additionally, the PROGRES specification consists of graph *transformations* that determine valid modifications of a graph, they provide the realized functionality. The graph transformations can be executed, to build up and modify a so-called host graph, representing the underlying data structure for modeling a certain application domain, here civil engineering. At this layer, *knowledge* and *design* are combined inside the PROGRES specification as they are built up from a similar internal graph representation. *Integration* between knowledge and design is therefore already given. The PROGRES specification, as well as the extensions to the UPGRADE framework [3], are fixed by the tool developer.

The second layer encloses the runtime dynamic part of the system architecture. On the left hand side, a domain expert, usually an experienced civil engineer or architect, *formalizes* conceptual *knowledge* specific to a certain class of buildings. The domain expert is called *knowledge engineer*. Conceptual knowledge comprises restrictions and requirements concerning the organization and functionality of a building, e.g. the size of specific rooms or the allowed length of an escape route. The knowledge formalization part is again subdivided into

two layers, namely the definition of a domain specific knowledge *ontology* and, based hereon, the definition of design *rules*.

The *knowledge ontology* serves as a formal definition and classification of the relevant knowledge concepts for conceptual design. The knowledge ontology is specific for one class of buildings. For example, different room, attribute, and relation types are defined here. They serve as atomic components for the following specification of design rules. The definition of the knowledge ontology is done at tool runtime by a knowledge engineer and is not fixed in the implementation. As the definition of the ontology is already an essential part of knowledge formalization, it has to be done by a domain expert (not the tool developer). Furthermore, this formalization is an evolutionary process with many iterations, so that a fixed knowledge ontology definition would not suffice.

Based on the concepts, defined in the knowledge ontology, in a second step the actual domain knowledge in form of *design rules* is defined by the knowledge engineer. The ConDes approach provides a graph-based visual language for knowledge specification which especially allows to define attribute, cardinality and relation rules, complex relations, and parameterized rules in form of runtime depending expressions (see section 4.2). The effort of knowledge formalization only pays off if several projects for the same class of buildings are carried out.

The right side of the second layer of Figure 1 is devoted to the conceptual design of buildings. Here, a *design ontology* can be elaborated at runtime analogous to the knowledge ontology. The design ontology may be specific for a special company, group of architects, being users of a design tool. It has to be mapped onto the knowledge ontology. To support the architect we further provide the possibility to import the knowledge ontology.

The *conceptual design* is elaborated in form of a purely *graph-based* representation. Room sizes and positions are not yet considered, only the existence and relationships between functional entities are defined. In literature, these widespread sketches are called *bubble diagrams* [48].

An i*ntegration* document [49][50] between the *knowledge ontology* on the left hand side and the *design* ontology on the right hand side has to be defined, to determine the corresponding ontology elements. The integration document is essential for checking the consistency between the design *rules* and the conceptual *design* by graph-based analysis. These *Design Checks* identify restriction violations and inform the architect. The semantics of all rule types is formally defined [51]. A conceptual sketch is called consistent, if none of the rules formalized by the knowledge engineer is violated.

Finally, in the last layer the developed tools are depicted. The *graph-based tools* are derived from the PROGRES specification. The UPGRADE framework provides a reusable and extensible platform for executing such graph-based programs in a problem adequate representation. The complete tool construction process using PROGRES and UPGRADE is shortly presented in section 5.1.

For knowledge specification ConDes develops a graph-based visual application, used by the knowledge engineer, the *Knowledge Graph Editor*. This tool provides functionality to elaborate an object-oriented knowledge ontology and to define the actual domain knowledge in form of design rules. The knowledge engineer is supported by problem adequate views on the knowledge graph, by filters, and layout algorithms displaying the graph in a structured manner.

For conceptual design we developed a second tool, the *Design Graph Editor*. This graph-based CAD tool allows to elaborate conceptual sketches, corresponding to the defined design ontology. Using the Domain Knowledge Editor, architects can concentrate on the relevant functional entities, their attributes and interrelationships.

By executing the graph-based *Design Checks*, inconsistencies between the knowledge specification and the conceptual design are discovered and visualized inside the Design Graph Editor to inform the architect. As we do not want to restrict the architect's creativity, we explicitly allow inconsistent sketches. The architect is free in his decision to stay in an inconsistent state, or to fix all identified errors.

The reader should notice, that for the right side of layer two and three of Figure 1 there is also some tool support based on industrial tools (bottom-up part, see section 1). The knowledge ontology and

design rules can be imported via XML-files to the tool ArchiCAD of GRAPHISOFT. ArchiCAD was furthermore extended by new concepts as *room objects* and *room links*. An architect may now use ArchiCAD to produce conceptual sketches. As above, they are checked against design knowledge to assure that they are consistent. The bottom-up tools are not described in this paper (see [5]).

4. A Visual Language for Knowledge Formalization

In this section, a novel visual *language* for *knowledge specification* is described. It corresponds to the left part of layer 3 in Figure 1 (Design Rules). The ConDes project follows a *dynamic* knowledge formalization approach, i.e. a domain expert, an architect or a civil engineer, should be able to formalize his personal domain knowledge. In a fixed approach, the domain knowledge would be formalized by the tool developer within the source code.

In the ConDes scenario, knowledge *formalization* as described is done in *two steps*: First, the basic concepts have to be defined in a domain ontology. Based on this conceptualization [52], predefined design rules can be used in a second step to insert relevant knowledge. Ontology as well as rules are specific to a class of buildings. Both are to be used for any project of the corresponding class.

As the dynamic knowledge definition is an essential part of the knowledge formalization process, a domain *specific* knowledge *ontology* is introduced as an *example*. The purpose of this example is to explain the expressiveness of the knowledge definition. A complete description of the realization of the ontology and the underlying graph-based mechanisms can be found in [4][8]. The knowledge formalization approach is based on object-oriented concepts. So, UML-like [53] diagrams are used for the representation.

4.1. *The Domain Knowledge Ontology Part*

The knowledge engineer develops the knowledge ontology thereby defining and classifying the relevant concepts. In the conceptual design approach, the domain *knowledge* ontology makes use of *three basic elements*:

- **Semantic objects** describe the conceptually relevant functional entities of architectural design,
- **Relations** describe connections between semantic objects,
- **Attributes** describe necessary equipment or properties of a semantic object.

The ConDes approach supports the knowledge engineer by providing *predefined fundamental concepts* for conceptual architectural design. These concepts, building site, building, storey, area, room, and section have *invariant* semantics, independent from the class of buildings. They, as well as their aggregation relation, are fixed in the ontology model part of the PROGRES specification (see layer 1 of Figure 1). A room is defined as an element enclosed by walls, a section is a part of a room, and an area is an aggregation of several rooms or areas.

The dynamic ontology can be structured by the knowledge engineer using aggregation and inheritance. The *inheritance relationship* between semantic objects is expressed, as usual, by an arrow with a white head. In Figure 2 one can see a cutout of a sample knowledge ontology, specific to car garages. The middle part contains the definition of the semantic objects. The semantic object *Room* is the root class of all rooms. It has three subclasses, *Motor Vehicle Room*, *Customer Room*, *Sanitary Room*, and so on.

The *aggregation relationship* between an area and several rooms is represented by a connection edge with a diamond at its beginning. In the example ontology, the *Customer's Toilet Area* is an aggregation of the men's, the women's, and the handicapped's toilet. In contrast to UML, no multiplicities are defined for the elements included inside the aggregation. Such facts are stored in form of special design rules (see section 4.2.5).

To express and define relationships between semantic objects, the knowledge ontology also contains *relations* (Figure 2, left hand side). Relations are used for user-defined relationships; inheritance and aggregation are predefined relationships. In the example knowledge ontology one can see three root relations: *Adjacency*, *Connection* and *Separa-*

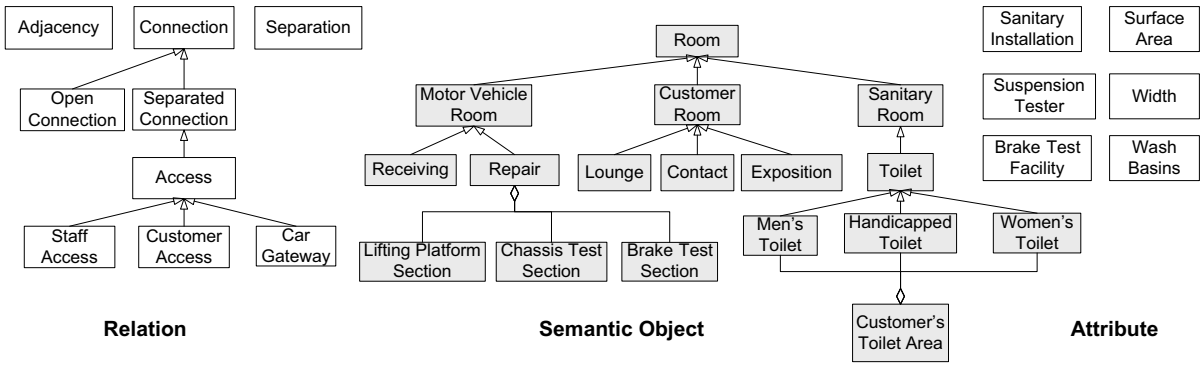

Fig. 2. Knowledge Ontology, specific for Car Garages

tion. While adjacency and separation express that two semantic objects should be neighbored and separated, respectively, the *Connection* relation is further specialized into an *Open Connection*, e.g. no wall in between, and a *Separated Connection*, e.g. a door. Finally, the separated *Access* connection is specialized into three further types of connections determining in which way it is allowed to be used.

On the right hand side of Figure 2, six *attributes* are defined. *Sanitary* is a Boolean attribute describing whether a semantic object should have a sanitary installation ("true" valued) or not ("false" valued). The *Surface Area* determines the size of a semantic object. Its allowed values are defined by an integer subrange.

As to be seen from Figure 2, there are ontology concepts which depend on a class of buildings, as *Car Gateway*, *Brake Test Section*, and *Brake Test Facility*. They are different from *Room* and *Connection*, which are specific for civil engineering but do not take care of a specific class of buildings. These *unspecific* parts of the the ontology belong to the layer 1 of Figure 1.

4.2. *The Design Rules Part*

Based on predefined or already available knowledge concepts (objects, relations, attributes) and predefined types of design rules, the knowledge engineer formalizes conceptually relevant domain knowledge. The goal is to put in all conceptual knowledge, specific to a class of buildings. Later, the *design rules containing* this *knowledge* can be used for automatic checks.

The *definition* of design *rules* is based on *two parts*. One part is runtime *dynamic* and consists of the elements defined in the domain ontology. The second part, namely the underlying design rule *schema*, is fixed in the PROGRES specification. It consists of a formal specification of the design rules.

The three *basic types of rules* are attribute rules, relation rules, and cardinality rules:

- **Attribute rules** enforce semantic objects to have particular properties, see section 4.2.2,
- **Relation rules** define the obligation or prohibition of the existence of certain relationships between semantic objects, see section 4.2.2, and
- **Cardinality rules** restrict the number of actual semantic objects of a certain class in the conceptual design, see section 4.2.3.

We now start by *explaining* the *basic types* of design rules and continue by stepwise introducing more powerful *further concepts*. The semantics of all design rules is explained considering some example rules, specific for car garages. The formal semantics of all types of design rules is defined in [51]. The operational semantic definition [54] forms the basis for the graph-based consistency analysis.

4.2.1. *Building up Design Rules*

To get an idea of the internal graph-based realization of the *design rules*, in this section their *built-up* is briefly explained. The description presented shows the connections between semantic objects, relations, and attributes in the graph schema in

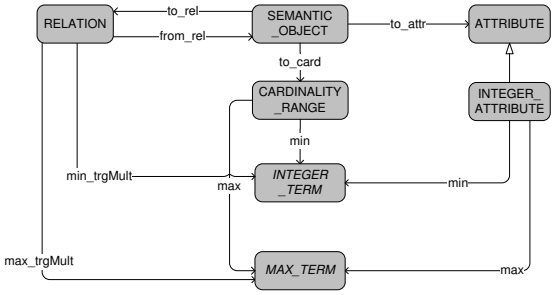

Fig. 3. Cut-out of Graph-Schema for Design Rules

a simplified form. The complete specification contains a number of further components, in order to be able to process the knowledge formalization and the consistency analysis.

In the PROGRES graph schema, the composition of design rules is formally specified. The graph schema fixes the static parts of graph-based applications. The dynamic aspects, so the actual program logic, is defined in PROGRES in form of graph transformations. For conceptual design support, each design *rule* is *composed* of one semantic object and, corresponding to the type of a rule, a relation with a second semantic object, an attribute, or a cardinality range.

A *cutout* of the *graph schema* for *design rules* is depicted in Figure 3. An attribute rule is here composed of a `SEMANTIC_OBJECT`-node connected via a `to_attr`-edge with an `ATTRIBUTE`-node. The constituent parts of a relation rule are two `SEMANTIC_OBJECTS`-nodes, interrelated with two edges, `to_rel` and `from_rel`, with a `RELATION`-node. Finally, a cardinality rule consists of a `SEMANTIC_OBJECT`-node and a `CARDINALITY_RANGE`-node, with a `to_card`-edge in between. In the *complete* graph schema, further nodes and edges are defined, to be able to reference a rule, to relate a rule to a complex context, and to handle the consistency analysis efficiently.

An important and repeatedly used concept for knowledge formalization is the *integer expression*. Integer expressions are needed to specify the values of an integer attribute, to define relation multiplicities or cardinality ranges. Looking at Figure 3, the `INTEGER_ATTRIBUTE` defines a range of values, valid for numerical attributes, e.g. a length between 4 and 6 m. The realization of this range in the graph schema is defined by an `INTEGER_TERM` as lower bound and a `MAX_TERM` for the upper bound. An integer term can be a constant integer literal or a complex integer expression. A `MAX_TERM` can either be an integer term or the star symbol, which allows arbitrary values. In the same way, minimal and maximal multiplicity restrictions of a relation and a valid cardinality range for semantic objects are formalized, using integer expressions based on `INTEGER_TERM` and `MAX_TERM`.

In the following, the *characteristics* of the visual *language* are introduced. For a better readability, the concepts are represented in a *condensed form* that gives an abstraction from the internal graph-based realization. However, all concepts are fully implemented, the knowledge formalization as well as the corresponding consistency analysis.

4.2.2. *Attribute und Relation Rules*

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

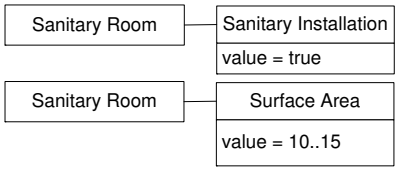

Fig. 4. Attribute Rules

As mentioned above, *attribute rules* determine valid properties of semantic objects in the design. There are *three types* of attribute rules which differ in the used data type. *Integer* attributes describe an interval of valid values, e.g. size of a room. *Boolean* attributes determine the availability of certain equipment, e.g. sanitary installation. Finally, *enumeration* attributes define a set of valid string values, e.g. orientation of a room to "north", "south", "west", or "east".

In Figure 4 two *example attribute rules* are depicted. The first, a Boolean attribute rule, demands each *Sanitary Room* to have a sanitary installation. The second integer attribute rule re-

stricts the ground area of such a room to be in between 10 and 15 sqm.

Relation rules determine the existence or non-existence of relationships between semantic objects. The number of semantic objects connected to a relation can be restricted. The declaration of *target multiplicities* allows a precise definition of how two semantic objects are interrelated. A target multiplicity greater than zero as *lower bound* demands each semantic object of the source class to be interrelated to at least the specified number of semantic objects of the target class. The *upper bound* restricts the maximum number of connected semantic objects. A special case forms the zero multiplicity for lower and upper bound. Such a relation rule prohibits the existence of a relation between the corresponding semantic objects.

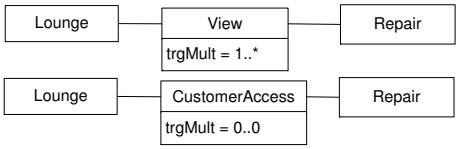

Fig. 5. Relation Rules

In Figure 5 two *example relation rules* are shown. The first relation rule demands the existence of a *View* relation between the *Lounge* and a *Repair* room. Thus, a customer can show the garage foreman the location of his car's problem. The multiplicity restriction only ensures that there has to be at least one possibility to look from the lounge to a car repair place. The upper bound is not restricted. The second relation rule deals with the customer access between a lounge and the car repair places. Here, the relation prohibits a customer access, expressed by the relation multiplicity set to 0..0.

4.2.3. *Cardinality Rules*

The number of occurrences of semantic objects is important for the organization of a building. We distinguish here, whether a semantic object is essential to guarantee the functionality of the building whether a semantic object can optionally occur, or whether a semantic object must not exist in the future building. A regulation is possible by defining the *lower* and *upper bound* of a *cardinality rule*. The lower bound determines if a semantic object is optional (= 0) or mandatory (>0), whereas the upper bound defines the maximum number of allowed occurrences of semantic objects. The upper bound is represented by a constant integer value or an integer expression, including the star symbol.

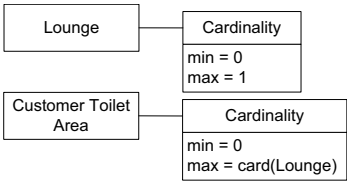

Fig. 6. Cardinality Rules

In the *example* of a car garage, a lounge is optional, and the maximal number of lounges is one. This fact is expressed by the first *cardinality rule* of Figure 6. Since a lounge is optional the "min" value of the cardinality range is set to "0"; the "max" value is set to "1". The second example cardinality rule of Figure 6 deals with the customer toilet area. Again, a customer toilet area is optional as the minimal number is "0". The maximal number, however, depends on the current number of lounges, which can be "0" or "1", as seen before. Thus, the semantics of this rule can be stated by "There can be one customer toilet area. If there are more, their number is at most the number of lounges". This rule already uses the concept of runtime dynamic expressions (see section 4.2.7 below).

4.2.4. *Inheritance and Aggregation*

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

The *analyses* and *structuring* of inheritance for knowledge specification is *realized* in the PROGRES specification by special nodes and edges. Complex graph transformations, operating on these graph elements, ensure the consistency of

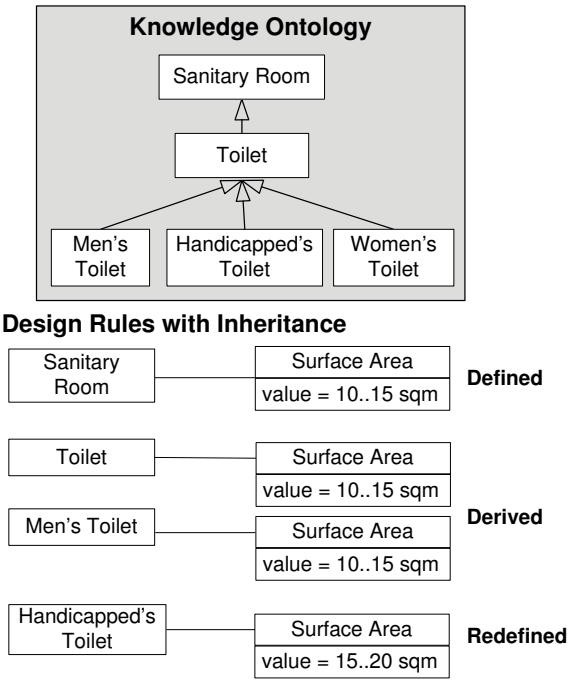

Fig. 7. Inheritance and Redefinition of Design Rules

these object-oriented structures. The internal realization, however, is hidden from the user.

The *example* in Figure 7 demonstrates the semantics of *inheritance*. The attribute rule for *Sanitary Room* demands the ground area to be between 10 and 15 sqm ("Defined" in Figure 7). Regarding the knowledge ontology cutout, one can see that the *Sanitary Room* is specialized into a *Toilet* which itself is specialized into particular types of toilets. The above defined attribute rule is now inherited to all subclasses ("Derived"). The attribute rule for *Handicapped's Toilet*, however, is redefined to guarantee a minimum turning circle for wheelchairs. All semantic objects inheriting from *Handicapped's Toilet* must therefore fulfill the redefined rule.

A further valuable concept of the knowledge ontology definition is the *aggregation* of semantic objects. Using the aggregation relation, complex semantic objects can be defined to be composed of a collection of simple or complex semantic objects. In contrast to the object-oriented classification mechanism, which describes a homogeneous set of semantic objects with similar properties, aggregation usually combines *heterogeneous* semantic objects. Constraints are relevant for aggregations as well. Therefore, the functionality is provided to define the basic types of design rules, namely attribute, relation, and cardinality rules, also for complex semantic objects. The *semantics* of *complex design rules* is slightly *different* from that of previously introduced simple design rules. For integer attribute and enumeration rules, we just demand that the rule has to be valid with respect to the aggregation concept (e.g. Customer Area). In case of Boolean attribute rules, we further distinguish between a positive or a negative application set by the Boolean value. If true, the expressed requirement for a Boolean attribute (e.g. Sanitary Installation) can be fulfilled by *one* of the semantic objects, encapsulated within the aggregation. Those Boolean attribute rules that forbid a certain equipment to exist, however apply to *all* encapsulated semantic objects. Thus, none of them is allowed to have this

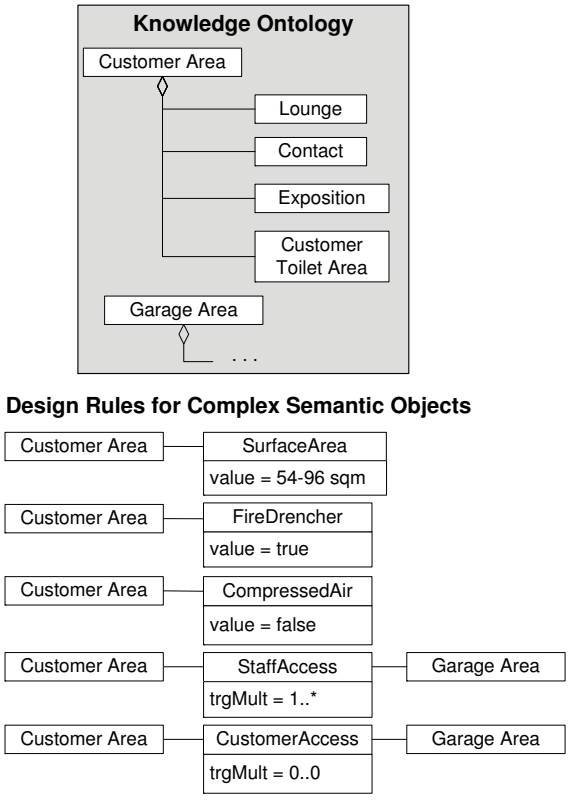

Fig. 8. Design Rules with Aggregation

equipment installed.

Regarding the example of Figure 8, a cutout of the knowledge ontology and three *example* attribute *rules* for a *complex semantic object* are depicted. A *Customer Area* is composed of a *Lounge*, a *Contact* room, an *Exposition* room, and a *Customer Toilet Area*. To restrict the surface area of this aggregation to be between 54 and 96 sqm, an attribute rule for the "Customer Area" is defined (Figure 8, first design rule). Simple analyses examine in this case, whether the sum of the surfaces of the aggregated semantic objects does not exceed the defined restriction. The second design rule of Figure 8 demands at least one semantic object of the aggregation to be equipped with a fire drencher. The third design rule prohibits to have compressed air supply in all semantic objects which are inside the customer area.

The previously introduced *relation rules* have also *extended semantics* if they are used for *complex* semantic *objects*. Analogous to Boolean attribute rules, the demand for a relation to exist between a complex semantic object and another can be fulfilled by one semantic object of the aggregation. The prohibition of a relation (target cardinality = 0..0) between complex semantic objects means the prohibition of a relation between corresponding component objects.

In Figure 8, two *example complex relation rules* illustrate the extended semantics. The first relation rule demands, that *Staff Access* is needed between the *Customer Area* and the *Garage Area*. This requirement is fulfilled if one of the semantic objects encapsulated inside the *Customer Area* has a staff access relation to one of the elements which are encapsulated in the *Garage Area*. Finally, the last relation rule in Figure 8 prohibits to sketch a customer access relation between any semantic objects encapsulated in the *Customer Area* and the *Garage Area*.

4.2.5. *Context Rules*

Up to now, design rules are valid for all specified semantic objects in the whole building, independent from their position. In practice, many *restrictions* concerning a conceptual design are *related* to the *context* of a room or an area, i.e. the enclosing semantic objects. To be able to formalize this contextual design knowledge, the concept of *context rules* is provided. A context rule is an extension to the previously introduced design rules, where the context only restricts the range of validity. In a conceptual design, the consistency analysis checks these design rules only in the scope of the defined context.

In Figure 9, some *example context rules* illustrate the expressive power. The sample knowledge ontology consists of a *Customer Area* containing a *Customer Toilet Area* which again contains a *Men's Toilet* and a *Women's Toilet*. A second area, the *Garage Area* is, among other things, composed of a *Men's Toilet*, *Women's Toilet*, and a *Hallway*. Obviously, different requirements and restrictions are valid concerning toilets inside a garage area or within a customer area. The first two context de-

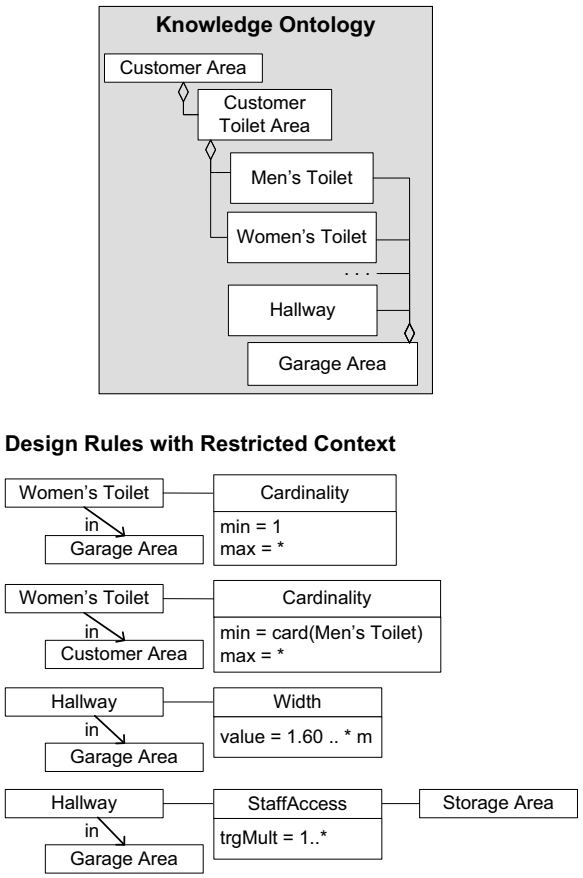

Fig. 9. Design Rules with Context Restriction

sign rules consider this fact. The fist context cardinality rule demands to have at least one women's toilet inside the garage area. The second design rule expresses the need to have inside a customer area at least the same number of women's as men's toilets. The card-operator returns in this case the number of men's toilets that are already sketched inside the customer area.

The third design rule in Figure 9 demands the width of the hallway to be at least 1.60 m, again restricted to those hallways that are inside the garage area. Further hallways, e.g. inside the customer area or in other parts of the building are not affected. Finally, the relation rule depicted at the bottom of Figure 9 expresses the need to have a *Staff Access* relation between this hallway and a storage area. Again, the scope is restricted to hallways inside the garage area.

4.2.6. *Complex Relation Rules*

As an *extension* to relation rules the visual language provides *complex relation rules*. They determine mandatory or forbidden intermediate relationships between two semantic objects. In conceptual design, composed relations are needed to express advanced concepts like transitive accessibility or restricted reachability. To enable a knowledge engineer to formalize such facts, this new type of design rules is provided.

A complex relation rule is realized in form of a *concatenation* of semantic *objects* and *relations*. The semantic objects at the beginning and the end of a complex relation definition restrict the application to connecting semantic objects of the determined class or its subclasses in the knowledge ontology. The inner components of the complex relation definition determine its internal structure.

An *example* of a *complex relation* rule is depicted in Figure 10. The definition of the complex relation *Indirect Access* demands that the access from one room to another contains an intermediate vestibule, e.g. to avoid noise pollution or bad smell. Below the definition of the complex relation in Figure 10 an *application* of this complex relation is given. To avoid a direct access between the lounge and a sanitary room, two design rules are defined. The first one, using the previously defined

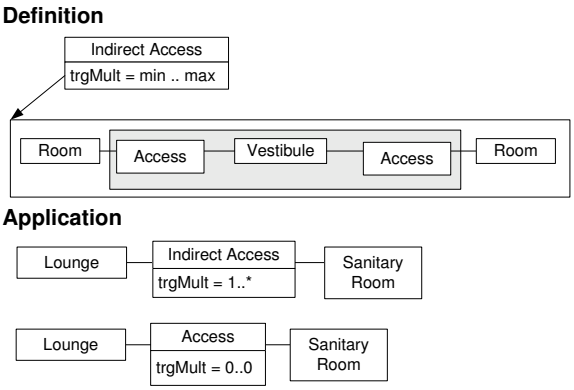

Fig. 10. Complex Relation Rules: Definition and Application

complex relation rule, demands an *Indirect Access* between the *Lounge* and the *Sanitary Room*. Additionally, the second rule, a simple relation rule, forbids the direct access between theses two semantic objects. Thus, all kind of toilets (see knowledge ontology) are only allowed to be accessible through an intermediate vestibule.

4.2.7. *Runtime Dynamic Expressions*

We previously introduced a cutout of the graph schema as the internal realization of design rules (Figure 3). We also briefly motivated the need for *runtime dynamic expressions*. They offer a powerful possibility, namely to define complex integer and Boolean expressions. Dynamic expressions are evaluated at runtime in connection to an actual building design. They do not introduce a new kind of design rule, but *extend* the *expressive power* of the visual language. The concept of runtime dynamic expressions is fundamental, because it is applied to the definition of attribute values and relation multiplicities as well as to cardinality ranges.

At the top of Figure 11 the *graph schema* for runtime *dynamic integer expressions* is depicted. The root class `MAX_TERM` is used to define the upper bound of an integer range. An upper bound can be either an `INTEGER_TERM` or a "STAR" (unlimited). Lower bounds can only be integer terms, as the star symbol is not allowed to be used. An `INTEGER_TERM` is specialized to a `CONSTANT_INTEGER` to represent an integer literal, and a composite `INTEGER_EXPRESSION`.

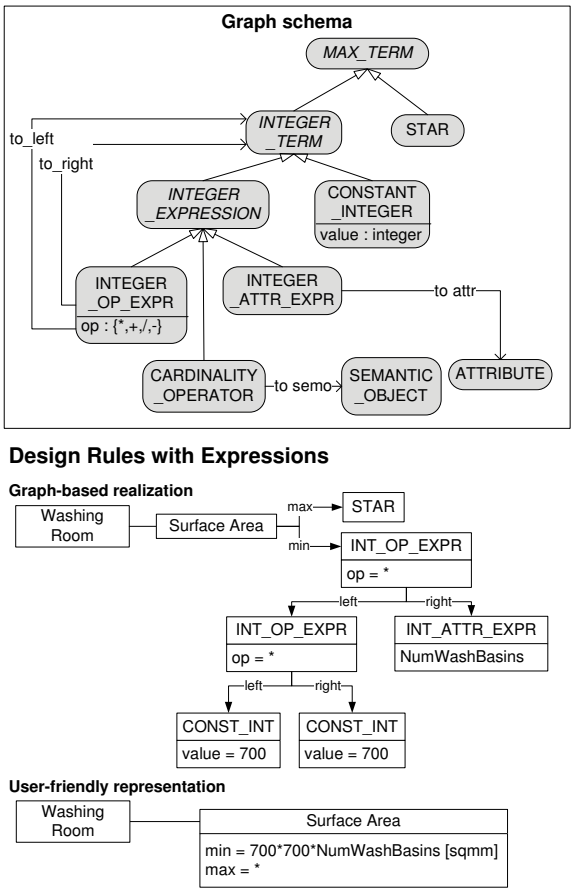

Fig. 11. Runtime Dynamic Expressions, Graph Schema and Example Rule

Three different *specialized integer* expressions are distinguished:

- **Integer operator expression** (`INTEGER_OP_EXPR`) represents an arithmetic expression, e.g. the multiplication of two integers. It consists of an arithmetic operator (*, +, /, -), and a left and a right operand of type `INTEGER_TERM`. The inductive definition allows arbitrarily nested operator expressions.
- **Integer attribute expression** (`INTEGER_ATTR_EXPR`) refers to an actual attribute value of a semantic object in the conceptual design.
- **Cardinality operator** (`CARDINALITY_OPERATOR`) computes the actual number of semantic objects of one class in the conceptual design (see Figure 6 for an example). In the graph schema the operator references a semantic object.

An *example application* of a runtime dynamic expression is shown in the lower part of Figure 11. The German law for working places (ArbStättV §35 (3)) demands a minimum surface area for washing rooms depending on the number of wash basins in the room. For each wash basin there must be a minimum ground area of 0.7 m times 0.7 m. Figure 11 illustrates how this regulation is expressed.

Instead of defining a simple attribute rule for *Washing Room* with static integer values, now the attribute *Surface Area* references two integer expressions for the lower and upper bound. As there is no need to restrict the size of a washing room, the upper bound of the attribute rule is determined by a "STAR" (unlimited). The *lower bound* is composed of runtime *dynamic* integer expressions. The `INTEGER_OP_EXPR` is calculated by the multiplication ("op = *") of the left term, again an `"INTEGER_OP_EXPR`, and the right term, an `INTEGER_ATTR_EXPR`. The value of the left term is again computed by the product of two `CONST_INTEGER` with value 700 mm. The integer attribute expression gets the number of wash basins which is represented by the value of the attribute `NumWashBasins` for the actual washing room. Using this information, the size of the washing room can dynamically be calculated.

The discussed representation shows the internal data structure to store the expression as a graph. The representation below is more readable and thus more user friendly. It matches the user interfaces representation.

5. Visual Tools for Supporting Conceptual Design

To prove the feasibility of the ConDes approach, *prototype versions* of different *tools* were implemented that support the definition of knowledge, the creation of conceptual designs, and analyze whether the specified rules are fulfilled. First, the construction process of these tools is described. Then, their usage is sketched.

5.1. *Tool Construction Process*

All implemented tools follow a similar tool construction process. As already mentioned, ConDes tools are based on the graph rewriting system PROGRES [2] and the UPGRADE framework [3]. Both tools have been developed at Department of Computer Science 3 of RWTH Aachen University in other projects. As PROGRES and UPGRADE are domain unspecific, many research projects from different domains (e.g. re-engineering of legacy telecommunication systems [55], data integration [50], project management [56]) are *using* this *tool construction infrastructure* for solving ambitious problems.

In PROGRES, the program logic and all stored data are given and processed in form of the general data structure *graph*. PROGRES is a high level specification *language* [57], all graph definitions and transformations can be defined in a *visual* and *declarative* way. Finally, PROGRES provides an *interpreter*, that allows to execute a specification and a *generator* for C-code.

The UPGRADE *framework* allows to create a *basic* visual *prototype* using the C-code that is exported by PROGRES. This prototype already provides some *generic base* functionality. Also, the representation of the graph nodes and edges is initially generic.

To provide a tool that is *customized* for knowledge specification and conceptual design, several *extensions* had to be made. First, different views were defined so that information can be represented in tables or trees, in addition to the graph representation. The graph view is provided with certain filters, that for each tool only a certain part of the entire graph is displayed. For graph nodes that appear in such a cutout of the graph, adequate node representations were defined, so that the user of the tool gets an easy and intuitive visualization. Additionally, layout algorithms are offered for positioning the graph nodes automatically to obtain clarity. The transformations specified in the PROGRES specification are not called directly by the user. Instead, all the functionality provided by the transformations is encapsulated within user-friendly dialogs.

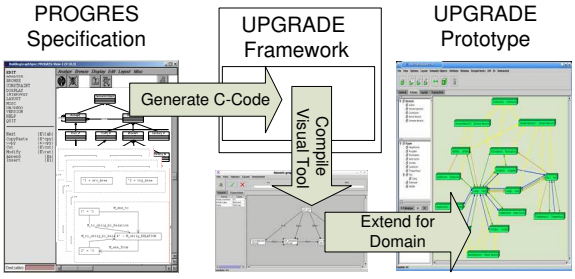

Fig. 12. Simplified Tool Construction Process with PROGRES and UPGRADE

Figure 12 illustrates the *tool construction process* with PROGRES and the UPGRADE framework. First, a formal PROGRES specification is elaborated. In the next step, PROGRES generates efficient C-Code. This code is then compiled and embedded in the UPGRADE framework, to obtain a graph-based visual application. Finally, this application is extended to the needs of a certain application domain.

This tool construction process is applied to all tools of section 3 following the top-down part of ConDes. Specifically, it is used for the ontology definition tool, the rule definition tool, the design ontology tool, the conceptual design tool, as well as for the checks to be performed.

5.2. *Knowledge Formalization Functionality*

First, lets take a look at the *functionality* of the *knowledge formalization* part. As introduced, knowledge formalization is done in two steps. It begins with the development of a domain ontology as basis for the definition of design rules.

The screenshot in Figure 13 depicts the *Domain Ontology Editor* used by a knowledge engineer. In this tool, semantic objects (e.g. rooms), relations (e.g. access), and attributes (e.g. width) can be defined. These ontology elements can be related to each other by inheritance and aggregation relations. In the screenshot, the *semantic object view* is shown in the upper half and the *aggregation view* in the lower half of the editor window.

The *ontology* defined in the *example* of Figure 13 is related to the previously introduced example ontology. The ontology is again specific for car garages. In the screenshot one can identify e.g.

the definition of the concepts *Customer Room*, *Sanitary Room* and further concepts derived from them. The definition of other semantic objects, e.g. areas and sections as well as the definition of relations and attributes is done similarly. Looking on the aggregation view in the lower part of the screenshot, one can see that the *Customer Toilet Area* is composed of four semantic room objects, the *Handicapped Toilet*, the *Men's Toilet*, the *Women's Toilet*, and a *Vestibule*.

By choosing one of the tabs at the top or bottom of the semantic object view, the knowledge engineer can *change* to the corresponding *view windows* to other semantic objects (e.g. area, section), relations, and attributes.

The next screenshot of Figure 14 contains the *Domain Knowledge Editor*, which is again used by a knowledge engineer, now to define design rules. This tool also supports the engineer to navigate through the domain knowledge by providing a visual, compact, and concise representation for the formalization of conceptual design knowledge. The design rules specified herein will be checked against an actual building design later.

The Domain Knowledge Editor consists of *two* main *parts*. At the left side, some *tree views* show the previously defined ontology elements as support for the knowledge engineer. In the main part of the Domain Knowledge Editor, the design *rules* are shown in a visual representation. The screenshot in Figure 14 already comprises some of the

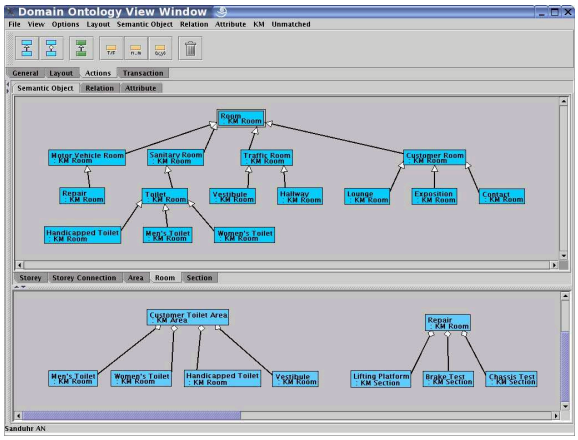

Fig. 13. Screenshot of Domain Ontology Editor

design rules that were introduced in section 4.2.

The first rule introduced in Figure 4 restricted the surface area of a sanitary room to be between 10 and 15 sqm. The *attribute rule* is represented in the Domain Knowledge Editor by showing the attribute *Surface Area* inside the node representing the *Sanitary Room*, together with the specified bounds of 10 and 15 sqm. The arrow, next to the attribute which is pointing to the right, indicates that this attribute rule is newly defined for sanitary rooms and is not inherited. Corresponding to the Domain Knowledge Ontology, the attribute rule is inherited by the concept *Toilet* and its subclasses. In the editor, an inherited attribute rule is represented by an arrow pointing to the top.

The *relation rule* introduced in Figure 5 demanded a view relation between the *Lounge* and the *Repair* place. Furthermore, the *Customer Access* from the *Lounge* and the *Repair* place was prohibited by a second relation rule. This relation rules are represented by *edges* in the Domain Knowledge Editor with the name of the relation in the middle of the line. The edge with a simple arrow at its end shows that e.g. a view relation is *demanded* between a lounge and a car repair room. The multiplicities for this relation (1..*) are also shown at the end of the edge. The relation rule is directed, therefore the arrow is only at one end of the edge. The second edge with a circle at its end indicates that it is *forbidden* to sketch a customer access between the lounge and a car repair room. This relation rule is *undirected*, it forbids the customer access in both directions. In the Domain Knowledge Editor, undirected relation rules are represented by edges with arrows or circles, respectively at both ends. Internally, an undirected relation rule is always realized by two *directed* relation rules. Because the multiplicities are in case of forbidden relation rules always the same (0..0), they are not shown within the editor. The dialog window in the lower part of the screenshot illustrates how a relation rule can be *inserted*. The knowledge engineer can use five tabs to exactly define and comment the inserted design rule.

The third type of design rule type, the *cardinality rule* was explained when discussing Figure 6. This design rule demanded that there may be a lounge, but maximally one of them. In the Domain Knowl-

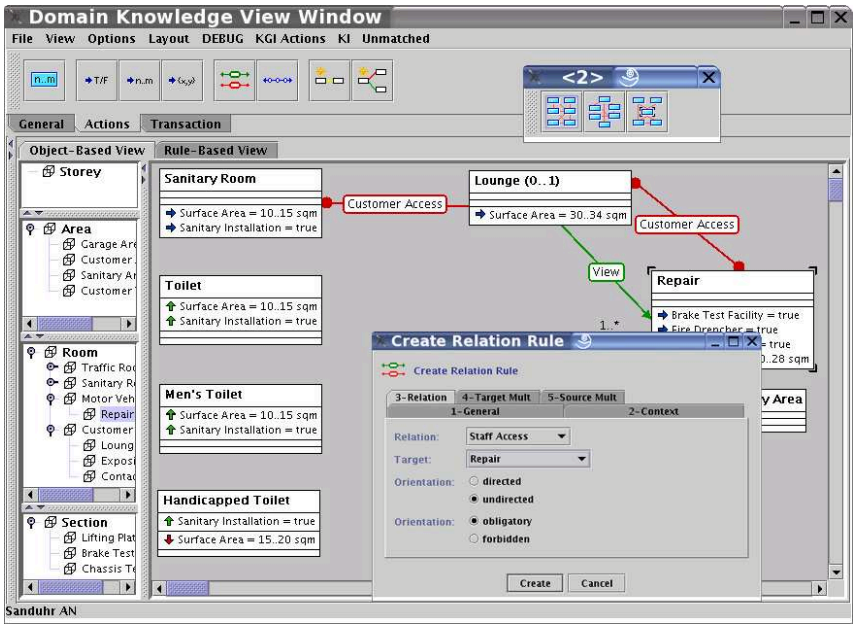

Fig. 14. Screenshot of Domain Knowledge Editor with Basic Design Rule Types

edge Editor, this cardinality range is depicted inside the node representing the *Lounge* on the right hand side in the title row.

Up to now, basic design rules and some *extended* concepts like inheritance of attribute rules were demonstrated. The second screenshot of the Domain Knowledge Editor, depicted in Figure 15 now demonstrates some of the *advanced concepts*.

To help the knowledge engineer reviewing the inserted design rules, the tools provide the possibility to temporarily display all currently effective design rules. In the first screenshot in Figure 14, the direct *Access* between the *Lounge* and the general concept *Sanitary Room* was forbidden. In Figure 15, all *inherited relation rules*, i.e. between the *Lounge* and all concepts derived from the *Sanitary Room* are displayed. For a better readability, these inherited relation rules are usually *hidden*. The inherited *attribute* and *cardinality* rules however are always *visible*, they are displayed in a condensed form inside the corresponding semantic object nodes.

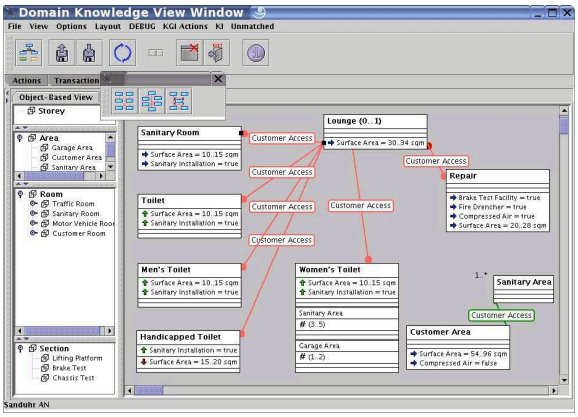

Fig. 15. Screenshot of Domain Knowledge Editor, with visualized Reference Rules

Looking at the screenshot of Figure 15 again, one can identify the application of a *context cardinality rule*. As demanded before in Figure 9, the number of needed *Women's Toilets* is depending on their position in the building. Inside the *Sanitary Area* there have to be at least three and at most five *Women's Toilets*. The number of *Women's Toilets* inside the *Garage Area* is however lower, here only one or two *Women's Toilets* have to be available. To display the context in a visual representation, several sections were inserted to the semantic object nodes. Each section then contains the attribute and cardinality rules which are effective in the corresponding context.

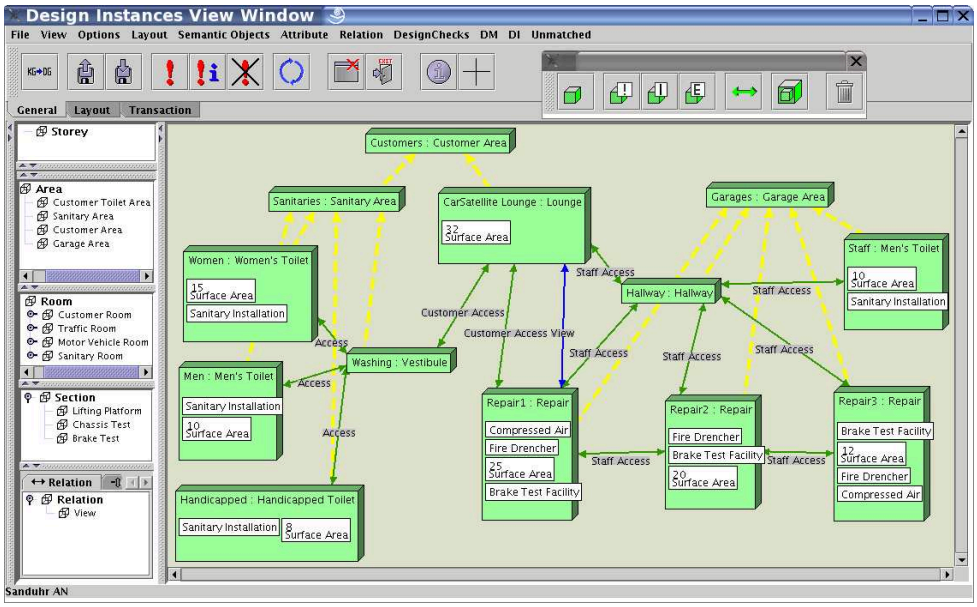

Fig. 16. Screenshot of Design Graph Editor, CAD tool for Conceptual Design

After finishing the knowledge formalization process, the knowledge engineer can *store* the domain *knowledge* in GXL [58], a standardized data format for storing arbitrary graphs. Furthermore, the layout of the knowledge graph can be stored, to help the knowledge engineer understanding the current state of the knowledge base when he *continues* to formalize design rules. Finally, a *documentation* of the formalized conceptual design knowledge can be generated, a documentation generator transforms all design rules and the inserted comments into a HTML page.

5.3. *Conceptual Design Functionality*

The tool for creating conceptual building designs, the *Design Graph Editor* is depicted in Figure 16. This tool is used by an architect in the conceptual design phase of a specific building design process. Using this tool, conceptual elements for the building elements and the relations between these elements can be defined. The representation is related to so-called bubble diagrams [48] that are extensively used by architects during the early phase of architectural design. Semantic objects in the Design Graph Editor are represented as rectangles and not as circles, trivially the level of abstraction is the same.

In the *example* depicted in Figure 16, a first simple *sketch* of a car garage is shown within the screenshot of the Design Graph Editor. The architect defined a lounge with a surface area of 32 sqm, shown in the middle of the graph view. Three toilets –a man's, a woman's, and a handicapped's toilet– are accessible from this lounge by crossing a vestibule. In addition to the definition of the surface area of these rooms, some of the needed equipment, e.g. sanitary installation, is already planned. The accessibility of the toilet area is restricted to customers.

The garage area is depicted in the right hand side of the conceptual sketch. Three different car repair places are accessible from the staff. Also, a men's toilet inside the garage area is accessible from the garage staff. The surrounding areas, the *Sanitary Area*, the *Customer Area*, and the *Garage Area* area are depicted at top of the screenshot. All included semantic objects are pointing to them to indicate their relationship.

On the right side of Figure 17, an alternative representation, the so called *boxed layout* is shown. Here, all rooms are layouted and positioned inside

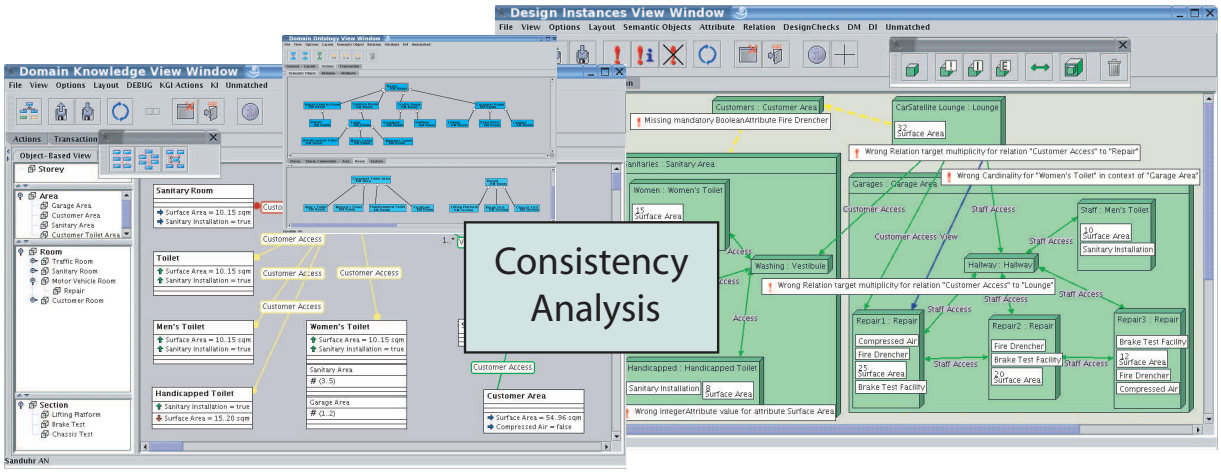

Fig. 17. Screenshot of the Knowledge Editor (left) and the Conceptual Design Tool (right)

the surrounding areas. This representation is similar to bubble diagrams. The conceptual sketch of a car garage helps the architect to get a first idea about the building components and their relationships.

5.4. *Design Checks*

The conceptual design tools provide the functionality to *check* a conceptual building *sketch*. This means that the sketch developed by an architect is checked *against* the design *rules* specified by a knowledge engineer.

The *example* sketch described in the previous section contains some *rule violations*. First, the handicapped's toilet has a surface area of 8 sqm, which is too small. In the knowledge definition a rule exists that demands a surface area between 15 and 20 sqm for handicapped's toilets. Second, in the conceptual design, customer access is allowed between the lounge and one of the car repair room's. According to the defined rules, no customer access is allowed between a customer room and the car repair room. As the lounge is a customer room, this rule is also violated. A further design rule demands to have at least one womans's toilet inside the garage area. In the conceptual design, there is only a men's toilet inside the garage area, the women's toilet is missing. Finally, there is no fire drencher installed in any room contained in the customer area.

The rule *violations* are *visualized* in the conceptual design editor by boxes containing an exclamation mark. A meaningful and more informative error *message* can be displayed optionally. The error messages are automatically positioned next to the element that caused the restriction violation.

The rule violations visualized are *advices* to the architect to review the conceptual design at the highlighted parts. It is not mandatory for the architect to eliminate all notifications and warnings in the design, as we do not intend to constrain the creativity and personal responsibility of architects. These notifications are rather to be understood as *hints* to *problems* that might arise later when the designed building is actually to be built and the restrictions from legal, economical, and many other domains are not met.

6. Summary and Future Plans

In this paper we introduced an expressive visual knowledge specification language for conceptual design in civil engineering. Based on graph technology, we described a dynamic knowledge model and a possibility to define design rules. We further presented the expressive power of the visual language by means of examples. The expressiveness of the three basic design rule types (attribute, relation

and cardinality rules) is extended by the concepts of inheritance and aggregation as well as by runtime dynamic expression and complex relations.

The defined knowledge in form of design rules serves on the one hand as reference work for architects and civil engineers. A more powerful usage results from the graph-based consistency analysis checking a conceptual sketch. The internal realization of the consistency analysis, not presented in this paper, is specified in the form of graph transformation using the graph rewriting system PROGRES. The developed tool support provides a useful representation for knowledge engineers and architects and demonstrates the feasibility of the approach.

There are two main extensions to the current state of our project. The first *extension* is the enhancement of the expressiveness of design *rules*. Design rules are atomic at the moment and it is not possible to combine them to *complex* constructs. When a conceptual design is checked against these design rules, each of them has to be fulfilled. Rules with practical relevance, for example lawful regulations, are often *complex* in that they are composed of alternative, conditional, or negated rules. To represent complex rules in our system, the atomic design rules will be extended to complex design rules which are composed of other design rules using Boolean operators as connectors. By Boolean operators the knowledge engineer has intuitive means to represent and build-up complex design rules using our tool.

The second extension concerns the *modularization* of knowledge and the *integration* of knowledge modules. It is unrealistic to assume that the complete knowledge represented by the design rules is acquired and formalized by one single knowledge engineer and at once. So, it is our goal to open up the opportunity to store knowledge in different modules. Therefore, the domain ontology as well as the design rules will be modularized, reasonably subdivided according to knowledge sub-domains. Modules will be layered, so it will be possible to define some base knowledge about industrial buildings on the topmost layer and to define specialized knowledge, like knowledge for car-garages, on some lower layer. As a result of the modularization, the modules themselves keep small and clear.

The ConDes project aims at elaborating domain specific tool functionality. Currently, the approach is restricted to the domain of architecture. The visual language, as the basis for the explicit knowledge formalization, has been developed according to the restrictions and requirements existing for architectural design. To develop a support for other application domains, like mechanical or civil structural engineering, the knowledge formalization part of the project has to be fundamentally changed. Nevertheless, the system architecture (cf. Figure 1), the tool construction method, and the experiences gathered in the domain of architecture can be adopted.

7. Acknowledgements

The authors gratefully acknowledge the support of this project by the German Research Foundation (DFG) within the scope of the priority program "Network-based Co-operative Planning Processes in Structural Engineering" [59]. Furthermore, the authors would like to thank the Nussbaum GmbH, especially Gerd Schneider, for fruitful discussions and preparing a collection of design rules. Finally, we thank A. Borkowski (IPPT PAN Warsaw) and A. Schürr (TU Darmstadt) for their valuable contributions to the CoDes-project within a joint research cooperation.

References

[1] M. Nagl, Ed., *Building Tightly Integrated Software Development Environments: The IPSEN Approach*, ser. LNCS. Springer, 1996, vol. 1170.

[2] A. Schürr, "Operationales Spezifizieren mit programmierten Graphersetzungssystemen," Dissertation, RWTH Aachen, Wiesbaden, 1991.

[3] B. Böhlen, D. Jäger, A. Schleicher, and B. Westfechtel, "UPGRADE: A Framework for Building Graph-Based Interactive Tools," ser. LNCS, A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, Eds., vol. 2505. Barcelona, Spain: Springer, 2002, pp. 270–285.

[4] B. Kraft and N. Wilhelms, "Visual Knowledge Specification for Conceptual Design," in *Proc. of the 2005 Intl. Conf. on Computing in Civil Engineering (ICCC 2005)*, L. Soibelman and F. Pena-Mora, Eds. ASCE (CD-ROM), 2005, pp. 1–14.

[5] B. Kraft and G. Schneider, "Semantic Roomobjects for Conceptual Design Support: A Knowledge-Based Approach," in *Proc. of the 11th Intl. Conf. on Computer Aided Architectural Design Futures (CAAD Futures '05)*, B. Martens and A. Brown, Eds. Heidelberg: Springer, 2005, pp. 207–216.

[6] Graphisoft, "ArchiCAD 8.1," www.graphisoft.com/ (05. 09. 2006), March 2005.

[7] B. Kraft and M. Nagl, "Parameterized Specification of Conceptual Design Tools in Civil Engineering," in *Proc. of the 2nd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE 2003)*, ser. LNCS, J. Pfalz, M. Nagl, and B. Böhlen, Eds., vol. 3062. Heidelberg: Springer, 2004, pp. 90–105.

[8] B. Kraft and D. Retkowitz, "Graph Transformations for Dynamic Knowledge Processing," in *Proc. of the 2006 Intl. Conf. on System Sciences (HICSS 2006)*, E. Robichaud, Ed. IEEE Press, 2006, pp. 1–10.

[9] B. Kraft and M. Nagl, "Semantic Tool Support for Conceptual Design," in *Proc. of the 4th Intl. Symp. on Information Technology in Civil Engineering (ICCC 2003)*, I. Flood, Ed. ASCE, 2003, pp. 1–12, (CD-ROM).

[10] B. Kraft and N. Wilhelms, "Interactive distributed Knowledge Support for Conceptual Building Design," in *Proc. of the 10th Intl. Conf. on Computing in Civil and Building Engineering (ICCCBE-X)*, K. Beucke, B. Firmenich, D. Donath, R. Fruchter, and K. Roddis, Eds. Bauhaus-Universität Weimar, 2004, pp. 1–14.

[11] Object Management Group, Inc., *The Common Object Request Broker: Architecture and Specification, Revision 2.6.1*, 2002, http://www.omg.org (14.6.2002).

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

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[14] J. Joedicke, *Entwerfen und Gestalten*. Karl Krämer Verlag, 1993.

[15] F. Steinmann, "Modellbildung und computergestütztes Modellieren in frühen Phasen des architektonischen Entwurfs," Dissertation, Universität Weimar, Weimar, 1997.

[16] G. Carrara and Y. E. Kalay, Eds., *Knowledge-Based Computer-Aided Architectural Design*. Cambridge, USA: Elsevier, 1994.

[17] R. D. Sriram, "Artificial intelligence in engineering: Personal reflections," *Advanced Engineering Informatics*, vol. 20, pp. 3–5, 2006.

[18] T. Maver, "A number is worth a thousand pictures," *Automation in Construction*, vol. 9, pp. 333–336, 2000.

[19] R. D. Coyne, M. A. Rosenman, A. D. Radford, M. Balachandran, and J. S. Gero, *Knowledge-Based Design Systems*. Boston: Addison Wesley, 1990.

[20] K. Meniru, C. Bédard, and H. Rivard, "Early Building Design using Computers," in *Proc. of the Conf. on Distributing Knowledge in Building (CIB w78 2002)*, P. Christianson, Ed. Dänemark: Aarhaus School of Architecture, Juni 2002.

[21] R. Mora, C. Bedard, and H. Rivard, "A Framework for Computer-Aided Conceptual Design of Building Structures," in *Proc. of the 10th Intl. Conf. on Computing in Civil and Building Engineering (ICCCBE-X)*, K. Beucke, B. Firmenich, D. Donath, R. Fruchter, and K. Roddis, Eds. Bauhaus-Universität Weimar, 2004, pp. 1–12.

[22] M. J. Sulaiman, N. K. Weng, C. D. Theng, and Z. Berdu, "Intelligent CAD Checker For Building Plan Approval," in *Proc. of the Conf. on Distributing Knowledge in Building (CIB w78 2002)*, P. Christianson, Ed. Dänemark: Aarhaus School of Architecture, Juni 2002.

[23] M. Eisfeld and R. Scherer, "Assisting Conceptual Design of Building Structures by an Interactive Description Logic-Based Planner," *Advanced Engineering Informatics*, vol. 17, pp. 41–57, 2003.

[24] D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, *The Description Logic Handbook*, 2nd ed., F. Baader, Ed. Cambridge: Cambridge Univeristy Press, 2003.

[25] M. Eisfeld, "Assistance in Conceptual Design of Concrete Structures by a Description Logic Planner," Dissertation, TU Dresden, Kassel, 2005.

[26] U. Flemming, *Case-Based Design in the SEED System*. Cambridge, USA: Elsevier, 1994, pp. 69–91.

[27] U. Flemming and S.-F. Chien, "Schematic Layout Design in SEED Environment," *ASCE Journal of Architectural Engineering*, vol. 1, no. 4, pp. 162–169, 1995.

[28] J. Szuba, A. Schürr, and A. Borkowski, "GraCAD – Graph-Based Tool for Conceptual Design," in *Proc. of the 1st Intl. Conf. on Graph Transformation (ICGT 2002)*, ser. LNCS, A. Corradini, H. Ehrig, H.-J. Kreowski, and G. Rozenberg, Eds., no. 2505. Springer, 2002, pp. 363–377.

[29] E. Grabska and W. Palacz, "Floor Layout Design with the Use of Graph Rewriting System PROGRES," in *Proc. of the 9th Intl. Workshop of the European Group for Intelligent Computing in Engineering (EG-ICE)*, M. Schellenbach-Held and H. Denk, Eds. Düsseldorf: VDI Verlag, 2002, pp. 149–157.

[30] J. Szuba and A. Borkowski, "Graph Transformations in Architectural Design," *Computer Assisted*

Mechanics and Engineering Science, vol. 10, no. 1, pp. 93–109, 2003.

[31] J. Szuba, "Graphs and Graph Transformations in Design in Engineering," Dissertation, Darmstadt University of Technology, 2005.

[32] G. Stiny and J. Gips, "Shape Grammars and the Generative Specification of Painting and Sculpture," *Information Processing*, vol. 71, pp. 1460–1465, 1972.

[33] U. Flemming, "More than the Sum of Parts: The Grammar of Queen Anne Houses," *Environment and Planning: B*, vol. 14, no. 3, pp. 323–350, 1987.

[34] R. Davis, H. Shrobe, and P. Szolovits, "What is a Knowledge Representation?" *AI Magazine*, vol. 14, no. 1, pp. 17–33, 1993.

[35] R. Fruchter and P. Demian, "Knowledge Management for Reuse," in *Proc. of the Conf. on Distributing Knowledge in Building (CIB w78 2002)*, P. Christianson, Ed. Dänemark: Aarhaus School of Architecture, Juni 2002.

[36] S. J. Fenves and J. H. G. Jr., "Knowledge based standards processing." *AI in Engineering*, vol. 1, no. 1, pp. 3–14, 1986.

[37] G. Schmitt, *Architectura et Machina – Computer Aided Architecural Design und Virtuelle Architektur*. Wiesbaden: Vieweg, 1993.

[38] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, pp. 34–43, 2001.

[39] S. Powers, *Practical RDF*. Sebastopol, CA, USA: O'Reilly, 2003.

[40] G. Stumme, "Formal Concept Analysis on its Way from Mathematics to Computer Science," in *Proceedings of the 10th Intl. Conference on Conceptual Structures*, ser. LNCS, vol. 2393. Springer, 2002.

[41] J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine*. Boston, MA, USA: Addison-Wesley, 1984.

[42] W. Kollewe, M. Skorsky, F. Vogt, and R. Wille, "TOSCANA – Ein Werkzeug zur begrifflichen Analyse und Erkundung von Daten."

[43] J. T. Nosek and I. Roth, "A Comparison of Formal Knowledge Representation Schemes as Communication Tools," *Intl. Journal of Man-Machine Studies*, vol. 33, no. 2, pp. 227–239, 1990.

[44] J. Sowa, *Priciples of Semantic Networks*. San Mateo: Morgan Kaufmann, 1991.

[45] E. Y.-L. Do and M. Gross, "Thinking with Diagrams in Architectural Design," in *Artificial Intelligence Review*, ser. 15, no. 1/2. Kluwer Academic Publishers, 2001, pp. 135–149.

[46] At Last Software, "SketchUp," de.sketchup.com (30. 08. 2006), August 2006.

[47] KollabNet Corporation, "KollabNet," www.kollab-net.com (30. 08. 2006), August 2006.

[48] E. Neufert and P. Neufert, *Architects' Data*, 3rd ed. Oxford, Great Britain: Blackwell Science, 2000.

[49] M. Lefering, "Integrationswerkzeuge in einer Software-entwicklungs-Umgebung," Dissertation, RWTH Aachen, Aachen, 1994.

[50] S. M. Becker, T. Haase, and B. Westfechtel, "Model-based a-posteriori integration of engineering tools for incremental development processes," *Journal of Software and Systems Modeling*, vol. 4, no. 2, pp. 123–140, 2005.

[51] B. Kraft and D. Retkowitz, "Operationale Semantik-definition für konzeptuelles Regelwissen," in *Proc. Forum Bauinformatik 2005*, L. Weber and F. Schley, Eds. Lehrstuhl Bauinformatik BTU Cottbus, 2005, pp. 173–182.

[52] G. Stumme and R. Wille, Eds., *Begriffliche Wissensverarbeitung*. Springer, 2000.

[53] M. Fowler, *UML konzentriert*. Addison-Wesley, 2004.

[54] M. Erwig, "Abstract Syntax and Semantics of Visual Languages," *Journal of Visual Languages and Computing*, vol. 9, no. 5, pp. 461–483, 1998.

[55] A. Marburger and B. Westfechtel, "Behavioural Analysis of Telecommunications Systems by Graph Transformations," in *Proc. of the 2nd Intl. Workshop on Applications of Graph Transformation with Industrial Relevance (AGTIVE 2003)*, ser. LNCS, J. Pfalz, M. Nagl, and B. Böhlen, Eds., vol. 3062. Heidelberg: Springer, 2004, pp. 202–219.

[56] A. Schleicher, "Roundtrip Process Evolution Support in a Wide Spectrum Process Management System," Dissertation, RWTH Aachen, Wiesbaden, 2002.

[57] A. J. Winter, "Visuelles Programmieren mit Graph-ersetzungssystemen," Dissertation, RWTH Aachen, Mainz, 2000.

[58] A. Winter, "Exchanging Graphs with GXL," in *Graph Drawing – 9th Intl. Symp., GD 2001*, ser. LNCS, P. Mutzel, M. Jünger, and S. Leipert, Eds., vol. 2265. Vienna, Austria: Springer, 2001, pp. 485–500.

[59] U. F. Meißner and U. Rüppel, "Homepage SPP 1103," http://www.spp1103.de/ (01. 01. 2006), 2006.