WORKING WITH PROTOTYPES: FROM CAD TO FLEXIBLE TOOLS FOR INTEGRATED BUILDING DESIGN

A. DRACH, M. LANGENEGGER and S. HEITZ
Institute for Industrialized Building Production
Department of Architecture
University of Karlsruhe, Englerstraße 7, 7500 Karlsruhe
Germany

ABSTRACT. The formulation of design knowledge as concepts, goals and rules cannot be captured in fixed and valid statements. The dynamic modeling of concepts and goals is, on the contrary, part of the design process itself. Tools that effectively support architects in their design should therefore never use predesigned mechanisms, but must be definable interactively according to design specifications. We propose the concept of prototypes as a cognitive model to represent and structure design knowledge. Prototypes incorporate an individual view of design in a systemic and organizational model for a defined area of interest. They actively control and guide design processes in supporting the organizational concepts for solutions. The zsTool implements these concepts on the basis of a modelling language. It provides a dynamic toolkit and user interface to support design as well as knowledge modeling.

1. Introduction

"The most successful designs are not those that try to fully model the domain in which they operate, but those that are in alignment with the fundamental structure of that domain and that allow for modification and evolution to generate new structural coupling" (Winograd and Flores 1986, p. 53).

CAD systems mainly support geometric modeling and drawing. Concepts to represent adequate product models and thus to enable integration and communication in a design process are still major issues in current research, and not only for building design. Our approach goes beyond product modeling in emphasizing the process of design supported by tools. To reflect and support the evolutionary aspects in this process, these tools have to combine fundamental structures for product modeling with an idea of design that is dependent on a designer's knowledge and interpretation.

The usual approach to provide decision support in design systems is to provide a set of knowledge bases that analyze and solve standard problems. This was the first approach of the authors when they participated in the development of the "Intelligent Design Tool Armilla" (Drach, Gauschel and Hovestadt 1990), a system that supported designers in the layout of complex service systems in buildings. Experience showed, however, that the problems of knowledge acquisition, extension and maintenance became more and more unsolvable with the development of larger design applications. Because of these problems, the systems remained limited in their scope and did not support complex design relationships.

The new approach has been fostered by a long period of experience in the development of expert systems and knowledge-based applications for building design. Our research involved investigations with a variety of tools and many activities in user-interface design. This practice and understanding of the problem provides the background to our proposal. The present concept of "zs" was developed in continuing research to provide a toolbox for applications in building design.
design. On that common platform, cooperating planners can configure their own specific tool to support different tasks like floorplan, structural or service-systems design.

![Diagram](image)

Figure 1. Scenario as Tool

The central idea of prototype configuration is based on a modelling language using semantic expressions, enabling designers to flexibly extend and change system behaviour. Interactive visual representations and editors support the parallel modelling of design solutions, specifications and control. The language incorporates new schemes that naturally describe semantic knowledge at different levels of abstraction as structural, geometric and functional aspects. The basic elements to define prototype knowledge are design elements, associations, shapes and constraints. Actions, events and queries supply the verbs and prepositions for design control in the formulation of active constraints. Facts and knowledge are not isolated, but treated as explicit and descriptive entities. Therefore, new environments can easily be built upon others, using and specifying existing versions. The s* system is developed as an object-oriented, extensible model and implemented in CLOS as a work-in-progress. Applications demonstrate design examples for conceptual service-system layouts.

2. The Problem of Design Integration

The growing complexity of building design, particularly through mechanical systems and building services, the shorter cycles of technical innovations and the overwhelming amount of different codes and regulations in the field of construction have created a situation where not only architects, but all parties involved tend to lose control of the design process if they do not use computer technology. These problems become vital with intelligent buildings and automated production facilities. The traditional separation of architectural and engineering tasks in building design, on the other hand, has brought on isolated software solutions and applications in the field. The different CAD systems and software applications are not able to communicate. This is not a technical problem of software engineering, but the problem of a common language. Product models could provide this common basis for communication but despite many efforts that have
been made (Björk and Penttiiä 1989), there is still disagreement about the capacity for expression and standardization of building representations, not only during the many stages of design and construction, but also for the different engineering contexts. This situation demands integration models and tools that incorporate the architect’s generalized view of design.

2.1. ARMILLA INTEGRATION TECHNIQUES

The ARMILLA research project started at the Institute for Industrialized Building Production in the early 80s, to deal with the integration problem in building design. It was a common belief that solving the problem required the combination of architectural knowledge and new developments in computer technology. The project’s formulation and results are based on the special architectural experience of Fritz Haller, who is best known for his designs of construction systems (USM Haller, MINI, MIDI, MAXI).

Figure 2. Installation ARMILLA, SBW Tiefengasse Lösenberg/ Münz CH
The ARMILLA installation model (Haller 1985) defines an integration technique based on geometric coordination and control, using spatial templates and different levels of abstraction. It had been developed as an abstract methodology to organize service systems in space and is applicable to any building design without the need for computers. The basic concept of the installation model is the structuring of space through a three-dimensional modular grid. Possible layouts and locations of pipes and ducts are restricted to certain routes or templates defined in the spatial grid. Different templates exist for different types and sizes of pipes. The modular and integrated organization of templates reduces possible conflicts between interfering systems to a minimum.

Since 1985 there have been efforts to apply the ARMILLA methodology to knowledge-based design environments. The results include several prototype expert systems to deal with the problem of automating service-system design (Mathis 1988; Raetz 1989). These systems, however, failed to represent and automate design solutions for problems of acceptable complexity. ARMILLA 2 (Gauchel et al. 1990; Drach, Gauchel and Hovestadt 1990) was the first approach to combine expert systems and CAD. It proved that the interactive approach was a promising concept to overcome the problems of brittleness and rule-explosion in complex and unstructured design tasks like building design. Such tasks not only require reasoning at different levels of abstraction, but also the integration of innumerable concepts from various domains and a great amount of common sense.

The work presented in this paper extends and combines earlier results in introducing a new level of flexibility and interaction in a model for supporting design tools. User control is suggested not only at the level of product modelling in finding appropriate design solutions, but also at the knowledge level, through interactive definitions for design consistency and support. Product modelling is only one part of the problem. Providing macro-languages to incorporate knowledge as part of a design is a major goal of our work. In this sense it is explorative in narrowing the gap between cognitive representations and software technology.

Corresponding research at the institute focuses on aspects of communication, cooperation and database technology in models for building design (Hovestadt 1991). Projects are funded by the government of Baden-Württemberg (Germany), CAD/CAM – Schwerpunkt, BMFT (Minister for Research and Technology, Germany) and DFG (German Research Association).

3. Expert Systems versus Flexible Tools

Despite the successful application of the ARMILLA model in expert systems, major questions of integration and representation of design knowledge remained unsolved. These problems concerned large and unstructured knowledge bases, knowledge acquisition and maintenance. Our experience and recent research in cognitive science supported various theories and goals for flexible knowledge environments. First, design knowledge cannot be captured in constant and valid statements. It is the individual property of a designer and must therefore be modifiable by designers. Knowledge is a lifetime process directly dependent on the person of expertise. The knowledge of an expert does not represent a defined optimized status of knowledge. Rather, it is subject to permanent changes, based on individual experience and learning (Becker 1992, p.47).

Secondly, complex and contradictory tasks require permanent interactive design decisions, supported by informational and assisting tools. Expert systems and knowledge-acquisition models tend to treat design knowledge as logically consistent expressions. This contradicts the architect’s view, in which goals and strategies are evolutionary concepts that evolve and change during the process of design. Architects have to take responsibility for their designs. Tools that support design must therefore be understood by the users. Thirdly, the permanent process of technological innovation in the building industry involves frequent changes in design methods and specifications. Easy adaptations and extensions of knowledge modules are therefore essential.
qualities for design knowledge bases. Finally, a modular representation of different aspects of design in the form of macro-languages could facilitate communication between distributed environments and users.

Expert systems have proved to support designers in solving clearly-defined standard problems (Kripilin et al. 1990; Lenart and Ketteler 1990). To date, however, applications have failed to effectively deal with complex and unstructured problem spaces. The question of structuring building design into standard problems that match standard solutions remains the main focus of present research and becomes even more complex for highly interactive design environments. Our proposal suggests a bottom-up development of decision-support systems to overcome the problem of blindness in expert-system applications. Design support in interactive environments may provide a basic level of control to inform users, if and how their solutions comply with defined goals and constraints. A user can then proceed with the design interactively while solutions are evaluated and checked for conflicts. On a second level, active knowledge can be added to constraints, proposing fixes for violations (Marcus and McDermott 1980).

Declarative and intuitive structuring of the supporting concepts and mechanisms is essential for users to be able to configure their own tool. According to classic expert system theory, applications are designed and maintained by knowledge engineers as domain-specific sets. Representation and formalism depend on the individual designs of knowledge engineers and are not visible to users, except through explanation control, which does not allow any rule modifications. The advantages of inference strategies in the separation of facts, rules and control tend to change into weaknesses with growing knowledge bases. Representations in blackboard systems lose their inherent relationships, which are evident and essential at the time of their formulation. Even experienced knowledge engineers admit that this problem makes large knowledge bases hard to maintain. On the other hand, advances in software technology have improved modularization-techniques for software applications. Object-oriented representations provide a more cognitive way to represent problems. Declarative representations of composite chunks of knowledge are more expressive and therefore easier to change and adapt to similar problems.

4. a+ Prototypes

The key concept and basic cognitive image for knowledge representations in a+ is the concept of a prototype. Prototypes are classification schemes for types of solutions, but their representation differs in some aspects from comparable concepts and terms used in software engineering. Prototypes aim at expressing information about different types of solutions, in the sense of giving an actual description of the solution spaces and interdependencies involved. In architecture and also in engineering, the term type or standard type refers to its synthetical nature. The pieces of knowledge it incorporates are already organized in relation to each other and coordinated in such a way that they do not conflict but have an integrating impact on the practical results. The type, as operational knowledge, may therefore guarantee coherence between an action and its goal (Muratori 1980).

An example of an architectural type is shown in Figure 3. All three floorplans of residential buildings by Frank Lloyd Wright follow the same organizational principle of a floorplan layout, despite their obvious variety in size and form. Their common prototypical concept could also be represented by a set of constraints that confine a solution to certain standards but do not restrict variations outside its range. We consider this concept as a basic cognitive model for design, which could be translated to all relevant areas and aspects. We therefore introduce it as a basis of our knowledge representation. We will show later how we use this concept to express design dependencies for service-system layouts.
4.1. WHAT IS A PROTOTYPE?

The following aspects may give a summarized definition of what is meant by prototype:

(i) Classification scheme for types of solutions. The language of architects and engineers already provides a variety of generally used types, to denote specific solution types for configurations in building or construction. This representation focuses on the descriptive power of prototypes in order to facilitate knowledge modelling. Prototypes may be organized in hierarchies, specializing certain compatibility and restriction rules for specialized systems.

(ii) Cognitive model of design. Prototypes represent individual categories for knowledge, incorporating aspects of context, time and experience in a single representational scheme. This notion follows the concept of "Gestalt" theory, where cognition is determined by the special structure and organization of information, assuming that the whole is more than the sum of its parts (Becker 1992, p.53). In representing information as a relevant section of the whole design, a prototype structures and reduces complexity. In this goal it corresponds to other concepts based on analogy like case-based reasoning (Hua et al. 1992) or shape grammars (Flemming 1987).

(iii) Synthetic, not operational model. We run into problems when we try to express our cognitive model in sequential operational activities. The information stored in prototypes is rather synethetical in nature. When designers think of prototypes, there is no operational model of how to generate a type of solution from a given situation. Nor is there a ready analogy for a situation, which only has to be mapped and adapted from a similar situation (case-based reasoning, routine design). In our representation, prototypes define the boundaries for their valid solution spaces, through specification of valid elements and configurations of elements, which are checked and organized when applied to a design situation.

4.2. THE MODELING LANGUAGE

Before we can illustrate how prototypes may be represented and integrated in an interactive design-support environment, we have to define a terminology for the problem space. To provide
a maximum of modularity in specifications, the modelling language is based on object-oriented modelling techniques (Booch 1991). It should therefore be no problem for users to specify their own objects from taxonomies and add control rules, instead of configuring and defining macros in a CAD environment. In order to describe prototypes in their area of interest and relevance for design solutions, we have to introduce an underlying level of description which we will refer to as the building-model level. This is the basic level of information and communication between prototypes. It could also be referred to as a product model of a building that provides an interface between cooperating applications in the process of building design, construction and maintenance, such as engineering, simulation, construction and facility-management tasks.

The following categories form our basic concepts with which to describe a design model of a building in all stages of design and in adequate semantics to serve as an interpretation background where prototypes can apply their knowledge in a useful way. Projects specify the actual building models for specific buildings. Design is not a linear process of refinement from the abstract to the more detailed. Rather, it progresses through recursive steps of revising, changing and refining decisions. A building model has to represent all levels of abstraction that were necessary to generate it. This is important to allow users and prototypes to step into and view through any level of abstraction at any time. Design levels form a structuring concept similar to views in filtering information in respect to certain abstraction levels. ARMILLA (Drach et al. 1990) proposes five major design levels: 1. spatial organization of the building structure, 2. requirement definition for service systems, 3. strategic layout design of service systems, 4. spatial coordination of piping routes, and 5. component specification and allocation. Design elements are the basic architectural concepts to describe structural, functional and geometric semantics in building design as systems, components and zones. Shapes constitute a separate language to model geometry, providing representation formalism for three-dimensional orthogonal structures and functionalities. These include expressing geometric hierarchies and relations, layout preferences, grid and functions, as well as queries about geometric relationships. A final category is associations: interpretations and conclusions about design states can rarely be deduced from the states of the elements alone, but mostly involve relations between elements. We therefore provide schemes to model user-defined associations to express semantic networks. More details concerning the modelling language are given in Sections 5, 6 and 7.

4.3. Working with Prototypes

We now turn to an example of how our theoretical concept of a prototype might guide a design and what kind of functionalities are provided when integrated in our interactive design system. Basically, a prototype represents more than just the cognitive concept or mental image a designer would use in exploring design problems. They represent a specific solution concept during a design session, through managing, controlling and organizing a defined space of interest (as a section of the whole state of design). Through managing elements in their relation to each other, it also defines the rules of consistency for associations between elements. We will assume that we have already worked out a layout for an air-supply system with our tool (see Figure 5) and would now like to be supported by prototype knowledge in the subsequent steps.

Installation. A prototype can be installed and used in a design situation interactively through selection menus and may interactively be removed or replaced by other prototypes. In order to install the knowledge only in the relevant section of the design, prototypes use their special installation knowledge. This means that they first check what kind of prerequisites have to exist, what kind of states and situations are relevant, which are the elements of interest, that should be controlled and organized, which elements can form a solution, and when is it appropriate to
warm, criticize, repair or make proposals. This type of knowledge is formulated in active constraints (see Section 7), which directly link prototypes to design elements.

Instantiation methods take care of the proper mapping of prototypes and elements to design situations defined in the building model, whereas attached rules and reactions supply assisting knowledge and procedures. They also incorporate knowledge about their appropriate design level and how to propagate themselves to other levels.

![Diagram of air supply system](image)

**Figure 5. Design example air supply system**

Interactive design. As long as a prototype is installed, its rules remain active. This means that they react to certain manipulations (events) in a protecting way, to keep the design consistent to a defined solution space. In case of violations, prototypes may either warn or criticize a user or make proposals and try to repair. In our example, the prototype for an air-supply system manages the relationships between a supply zone, the outlets and the ducts that connect an outlet to the riser for supply air. When an outlet or riser is interactively moved to another location in the building, the prototype will take care that the connecting ducts are adjusted. When a new zone is added or an existing one is resized, the proper layout for outlets may be generated or altered. If the size of a zone and its necessary air-supply volume extends the capacity of the connected riser, this information is brought up to the user, proposing several reactions like dividing the zone and introducing a new riser, or changing the riser's diameter if possible, which also increases its capacity.

Design conflicts. Constraints or goals in a prototype definition or between concurrent prototypes may contradict each other in trying to organize the same elements to meet their antagonistic goals. This is a very common case in design. You may, for example, state the rule, that risers are only allowed inside of shafts, but your HVAC prototype defines a maximum distance between outlet and riser. In some cases there will be no way to comply to both constraints. Then only an experienced designer can decide, depending on the overall situation, which rule can be relaxed. We therefore do not provide any automation to resolve these inconsistencies, like other approaches with truth maintenance systems. On the contrary: conflicting goals are inherent in the design process. The advantage of our approach is that they are made explicit and in that way can
be resolved interactively. The difference between an acceptable and an excellent design solution often lies in the decision which criteria should be relaxed in a certain situation.

4.4. Configuration of Prototypes

Instead of providing users with a predefined set of prototypes that represent a limited selection of solution types for a problem, an enforces the interactive redefinition and configuration of prototypes in order to incorporate individual requirements and knowledge. Any time during design modelling, the level of prototype configuration can be entered and existing prototypes can be modified in their functional behaviour or "semantic" interpretation of a problem. Prototypes are either defined from scratch or redefined and extended by inheriting elements from other prototypes.

User definitions. When using prototypes it often happens that their standard behaviour may be improper. Certain goals or technical constraints may be either obsolete or just inconvenient in the actual situation. Experiences have shown that even simple mechanisms can provide excellent design support when accompanied by a proper model for semantic control. A tracing mechanism for piping nets, for example, might determine that connected parts should be dragged along in order to keep the net together. This rule might be applicable only under certain conditions, or not at all, depending on the prototype's definition. The procedure can vary with different prototypes defined by different designers and their individual style of design.

Relaxations. Prototypes provide additional switches to enable relaxations. When turned off, interactive design can go on without warning. As soon as the switch is turned on again, the new situation is evaluated for restrictions. This applies to whole prototypes as well as to selected constraints.

Layers of information. A great problem, but also a challenge, for our approach is providing adequate user interfaces that allow informational transparency about a prototype's scope and its semantic and behavioural details as well as meaningful tools for its modification. Prototype configuration is structurally organized around layers of information (Figure 6). A complete view of the actual state of knowledge becomes visible through navigation along knowledge paths. Starting from any selected design element of the generated building model, a user can request and browse through information about its actual state (design view), including geometric and semantic properties, associations (association view) and related prototypes and constraints (prototype view).

Constraint rules. The best way to identify constraints, which are supposed to handle new conditions and behaviour for a prototype, is to concentrate on each class of elements involved and check for all kinds of roles and parts that may take in the solution. Forms facilitate the formulation of conditions. Queries support definitions in supplying a language to express complex search patterns. Constraints are triggered by events. In the conditional parts, design situations can be described by combining queries. Queries may not only relate to geometric information like distance or relative location but also concern type and association-specific states of design elements. In the reaction part, the results of queries can be linked with actions. A detailed description of constraints and queries is given in Section 7.
5. Entities and Associations

Marvin Minsky's theory of mind (Minsky 1985) captures many cognitive aspects of design. Most striking is his theory of k-lines and societies of k-lines. He distinguishes organizations of higher-level and lower-level agents. Each higher level of description must add to knowledge about lower levels rather than replace it. Our approach uses entities organized in object hierarchies and association networks to represent the notion of agents and agencies which are organized through k-lines. In the representation of entities and associations, we relate to languages for product modelling like STEP/Express (Andref 1992) and NIAM (Nijssen and Halpin 1989).

Design elements serve as the base class for entities. Design-specific entities may be built upon more general concepts that are provided with our modelling language. General concepts in building design include: (i) systems to represent structural entities in capturing hierarchies of functional decompositions into subsystems or components, which semantically relate to certain functional tasks in a building and have no geometric representation (i.e., structural system, circulation system, air-conditioning, cold-water supply), (ii) components as parts of systems, which form the end nodes in a design solution and, depending on the level of abstraction, may represent actual physical components and require geometric representations (i.e., partition, riser, duct, outlet), and (iii) zones that describe spatial organizations in buildings. They are important concepts to organize building hierarchies as semantic and spatial containers for components. Systems need zones to define spatial requirements for their functional tasks. Zones may overlap. Several zones representing different aspects of the design may even occupy the same space.
Examples include room, core, shaft, service zone, circulation zone, installation zone.

![Diagram](image)

**Figure 7. Design elements**

*Associations are complementary elements to entities in forming the k-line connections between design entities. They are important features in representing design, not only as structural solutions, but in describing intermediate steps or direct goals between corresponding entities. A similar approach has been used in ACCORD, a system for arrangement and assembly tasks that uses roles to denote valid states and goals between objects (Hayes-Roth et al. 1986). This holistic or coherent view reflects much better mental images in design than the operational view in rule-based representations, where agents form an unstructured mass of independent views.*

One representation uses the following definitions based on concepts introduced by Rumbaugh et al. (1991): roles define the direction of links between two or more entities. Each role has its name tag, denoting a proper identification of the semantic type of corresponding association. The language provides forms to build user-defined associations from basic concepts. Links create association networks during design (product modelling) through instantiating direct paths between objects. That way they provide an additional level of information in forming search patterns for problems. Because associations are often interrelated (as shown in Figure 8), the problem of keeping association networks consistent with defined semantics is an important issue.

![Diagram](image)

**Figure 8. Associations and roles**
In our approach this is managed at knowledge level (prototype specification) through user-defined constraints on associations, that watch for inconsistencies and use applied methods to restore semantic coherence.

6. Shapes

Geometric representations in a+ use shapes as a uniform description model to represent geometry in buildings (Langenegger 1992). In many CAD approaches, but also in shape grammars (Flemming 1987), geometry is the main focus for representation, geometric and semantic aspects of design are represented in a single concept. The shape model introduced with a+ proposes a more refined solution. Unlike traditional CAD systems, where geometry and visualization, as inseparable concepts, are the only representations for design, we propose the deliberate separation of semantic and geometric aspects. Shapes do not represent design semantics and therefore can never exist independent of design elements. They only form geometric properties of design elements which may be represented through one or more shape objects. These properties provide the design element with powerful representational schemes and functionalities. Shapes are defined by an enclosing box and a direction vector, and are designed to manage their own coordinates in a three-dimensional, orthogonal structure of space. They support interactive geometric manipulations as well as geometric queries on shapes and relations between shapes. The major representational schemes and functionalities of shapes are described below.

![Three dimensional representation of shapes](image)

Shapes are associated with design elements. Shapes represent geometric properties of design elements. They provide functionalities to query their associated elements and receive or propagate information about changes from the geometric (shapes) to the semantic (design elements) model.

Geometric hierarchies. Container shapes may include other shapes. This enables the structuring of shapes into purposeful hierarchies to reduce geometric search for a given problem. Relevant subsets of design elements may be grouped in meaningful containers to decompose a design in relevant sections. Building hierarchies and zones, for example, may geometrically be represented by corresponding shape hierarchies. Included shapes define themselves relative to the boundaries and coordination system of their parent. Methods are provided to query groups of shapes along hierarchies for defined criteria.
Geometric coordination is supported by three-dimensional coordination systems in the form of a complex grid. Specifications define the proper alignment of shapes in position, size and direction relative to the grid. This functionality allows the specification of abstract coordination models that manage possible locations of components and elements in a building. Higher-level shapes determine coordination for the next lower level. Shapes may align themselves in all three dimensions to rulers or zoning templates of their parent shape. The individual shape defines the details of alignment through methods that specify if and how to adjust to the grid, using additional information about adjacent shapes that might affect the proper layout. After modifications, all affected shapes are informed automatically to check and readjust their alignment to their surrounding coordination systems.

Layout preferences describe the proper size of shapes in all three dimensions. Shapes may be constrained in their length, width and height to minimal and maximal sizes, or dimensions may be defined in relation to each other. Values are automatically adjusted to follow these constraints after modifications of shapes. Layout preferences define the range of size and form of shapes; they do not specify possible positions and locations.

Geometric manipulation. An interface to model the distinct geometry of shapes provides methods to move, place, rotate and scale shapes or to change their location in geometric hierarchies. Changes are automatically propagated to associated shapes through hierarchies to enable their adjustment according to the changes.

Geometric queries. Shapes supply a query language to test geometric relationships between shapes. It includes basic functionalities to test the relative locations (north-of, adjacent, inside etc.) as well as comparisons (smaller, larger, distance, etc.). Primitive queries may be combined in expressions to test complex relationships.

7. Active Constraints

The concept of active constraints as the constituent elements of prototypes is introduced as the central structure to define and provide dynamic control in an interactive design environment.
(Heitz 1992). Constraints comprise flexible and powerful representational schemas for semantic specifications of prototypes and are designed to assist users during design in several ways.

![Diagram](image)

**Figure 12. Geometric coordination with shapes**

First, user-defined constraint rules check for semantic consistency of associations between systems, components or zones in a design; they supply methods for correction. For instance, the consistency rule for an import-output association between two components of a piping system may determine that both have to be members of the same subsystem. If one component is attached to a different subsystem during design, the improper links have to be removed. Secondly, user-defined or library constraints check if a design situation meets certain specifications. In case of a violation the same constraints can then supply users with detailed information about the problem. For instance, a restriction for waste-water systems may state that outlets can be placed only within a certain distance of a riser. If a user should move a riser or outlet outside these limits, the constraint may then pop up the information about the restriction.

Thirdly, constraints actively assist users in certain tasks through adding operational information about methods of possible fixes of the problem. For instance, in the previous example, the constraint could be extended by a modified reaction method, indicating several possibilities to fix the problem. This might include undo previous command, move riser, move outlet. Or the reaction might suggest a standard fix method, which could involve installing a new riser at the appropriate distance. Finally, constraint-directed search and optimization strategies can assist users in solving tedious configuration or allocation tasks. Reaction methods of rules are used to formulate knowledge-based correction rules following a propose and correct strategy (Marcus, Stout and McDermott 1988; Puppe 1990, p.143). For instance, a typical problem to which this strategy could be applied is the allocation of competing piping systems in their optimal spatial templates in a building, where many geometric and functional constraints may interact.

**Active constraints and prototypes.** Active constraints are always associated with prototypes. They may be linked to one or more user-defined prototype, which enables the easy definition of prototype variations. As soon as a prototype is installed in a user environment, all associated constraint rules are installed on their defined design elements. When a user interactively deinstalls the prototype again, all constraints are removed as well, unless they are not used by a second prototype which remains active. The association of constraints with prototypes defines prototype-specific sets of constraints. This provides a powerful instrument to structure constraints in
problem-specific units. Any time during design, a user may follow up which constraints are active with installed prototypes.

Figure 12. Prototypes and constraints

Constraint formulation. Constraints consist of a trigger part, a conditional part and an active part. Control is supplied through a kind of event trigger mechanism which directly connects design elements with constraints. This way inference can be provided without the overhead of a global matching algorithm. Another advantage is a more declarative structure of the knowledge base in the direct connections between design elements and constraints. The conditional parts of constraints represent situation patterns describing the valid solution space for the defined constraint. Queries are intermediate representations to describe conditional parts in constraints.

Constraint
trigger design element <system>
event: link <link-obj>
condition if and
<system> = of-type zoning air supply system
<component> = <system> is-linked-with-role has-components
<riser> = get-all <component> of-type riser
number-of <risers> greater-than 1
end and
reaction divide-system <system> = <new-system>
unlink <link-obj> with-role component-of
link <link-obj> with-role component-of <new-system>

The goal is to provide reusable templates for an easier description of situation patterns. Elements of queries are set definitions, predicates, type checking, logical operations, variables and the access to design elements, associations and geometric relations. A user may furthermore combine queries to define reusable macros for repetitive conditional expressions. The reaction part provides the same interface for actions on design elements that designers use through graphic manipulation tools. For example, a zoning air supply system may only be linked to one riser. The
constraint states that if a second riever has been linked (either interactively or by reactions), the system has to be divided.

Event trigger mechanism. Reactions of constraints invoke the same event trigger mechanism as top-level user commands. Design elements incorporate knowledge of how to transform actions into object-specific operations. Actions are linked to corresponding types of events. Events are created after the execution of an action and sent to a constraint handler. For each action-event cycle, the appropriate constraints are instantiated and sorted into a queue, which is processed while the conditional parts of the constraints are tested. In case of a violation, their active parts are called, which may invoke additional actions and events until the queue is completely processed. Despite the limits of this inference strategy, the idea of flexible constraints can be supported by the present implementation.

8. a+t-Tool

The a+t-Tool provides an interface and mechanism to use the concepts of prototypes, design elements, shapes and constraints in a system and demonstrates design support and assistance according to the ARMILLA model. Graphic editors and views are the major means of information and system interaction. Design control and assistance is enabled using event-driven inference based on object specifications. We provide a toolbox with a basic set of prototypes, typical design elements, associations and constraints, which can be used as is, or extended by the user. All relevant functionalities that form the interfaces to shapes, queries, actions, user-commands and visualizations are encapsulated and inherited from the basic concepts. Public protocols are supplied for these interfaces, based on the idea of a meta-object protocol (Kiczales et al. 1991), which allows users to adjust the design and implementation to suit their particular needs. In other words, users are encouraged to participate in the language design process. Applications are built as sets of additional domain-specific prototypes, design elements and associations. These sets can also be combined to easily reuse existing knowledge for new areas.

8.1. Visualization and Interaction

The interactive graphic user interface reflects the different layers of information already introduced as design view, association view and prototype view. Basic concepts for visualization and interaction are view objects, editors and commands. The design was strongly motivated and supported by the following goals: (i) direct interactive user control (any time during processing a user may control system behaviour according to his or her will), (ii) design elements are visualized as graphic symbols (a user may directly manipulate these symbols by mouse commands, similar to drawing tools like MacDraw), (iii) editors view and structure the building model of a project under certain aspects and relations, and (iv) editor tools support hypermedia connections and browsing along design views of a building model.

Design elements in a+t are not able to visualize themselves in the user interface; they use view objects as independent graphic-symbolic representations. View objects may represent different appearances, characteristics and information of design elements under different criteria like geometric, functional or semantic aspects. In providing concepts for symbolic, geometric or textual representations, they are specifically designed to handle the different kinds of visualizations of design elements. For each representation in an editor window, a design element uses a separate view object. Visualization, user interaction and communication of and with design elements is entirely managed by view objects. This is done by a communication protocol between both objects that allows view objects to query a design element for the relevant information it
needs for representation. Visualizations are constantly updated after changes on design elements to reflect the proper and actual state of the design.

Figure 13. A Tool: Geometric editor, prototype viewer and constraint violation

Editors provide a view on a defined portion of a project. Editors may, for example, visualize the geometries of design elements of a building in plan or section the way we are used to view a design on a CAD screen. The same situation could be represented as an association network in another window. For a+ we developed the abstract concept of an editor description to describe the various functionalities of different view concepts and to support the generation and redesign of editors. Editor descriptions include definitions for the scope of the editor as the relevant
portion of a building model, the form of visualizations and the supporting tools for manipulation. Filters define criteria for representation like type, location, relationships or other properties of design elements. For the filtered elements, an associated view object is generated and exposed in the editor. This way a building model can be viewed, checked and manipulated differently according to the editor's definitions.

Design elements in editors may be manipulated through manipulation tools. The underlying concept of any interactive manipulation is the 'a' command, which is performed by a user event in the form of a button or mouse command on a view object. The command may represent a series of actions, which are instantly propagated to the related design element where they invoke the appropriate operations. Design elements represent at any time the actual state of a project; their visual representations are only updated as a result of the operations.

Figure 14. asRule editor
8.2. User Interface

The a+Tool includes the following editors and tools which are modifiable to support interactive design in different environments: geometric editors represent geometric properties of a building model which can be viewed in plan, section or elevation at any scale. Tools are provided for three-dimensional geometric browsing and manipulation. Association viewers explore the semantic networks of associations. Associations and design elements may be viewed or manipulated graphically. Instance viewers display the actual state of design elements during design. The information is structured in appropriate textual windows and panels that enable browsing through all information and instances accessible in a project. Prototype tools show the state of installed prototypes in a design environment. Additional functionalities allow the user to view and relax or unrealized associated constraints. Prototype editors support designers in configuring or modifying prototypes and constraint rules. Design palettes display libraries of predefined design elements and prototypes. They may interactively be selected and copied (instantiated) to a project.

The current implementation of the a+Tool and language is in CLOS on DEC-3100 workstations using GINA (Spence and Beiklen 1990) as an interface toolkit. The implementation is designed to be portable to other object-oriented languages, or object-oriented database management systems and implemented as an extensible model.

9. Summary and Conclusion

The impetus for the a+model was formed by experiences in the field of building design integration. To date, architectural practice does not offer standard models to structure building design, nor does it provide any integrated design methodologies. The process of building design still appears as unstructured and orderless and is to a large degree dependent on human creativity and control. In this situation, we do not believe that tools should replace human control and communication, but tools could be quite efficient in facilitating it. Our proposal is centered on a user's perception of design, which means he or she should be enabled to structure his or her own environment within the limits of an integrating language framework.

The concepts developed in our model concentrated on the idea of providing public interfaces for design knowledge. Questions of data management, CAD integration, product modelling, concurrent design and multi-user environments, which are important issues in this field of research, were only touched upon by our approach.

The model has proved to be quite promising during implementation. In structuring and combining exchangeable knowledge modules with semantic and geometric representations, we could provide a useful model for the integration of user-defined environments and data bases in building design. The aim of our flexible tool is to point out perspectives. This aim might conform well with the next generation of CAD systems based on object-oriented database management systems. In some respects, this vision is not so different from using style formats in a word-processing system, instead of using prototypes in a CAD system. We think that knowledge modelling in design-support systems could become as easy as programming Hypercard or Excel.

References


