Area-Dependent Constraints for Design Control in a CAAD Environment

Peter C. Lockemann, Rose Sturm, Jutta A. Mülle
Institut für Programmstrukturen und Datenorganisation, Universität Karlsruhe, Germany

Volkmar Hovestadt
Institut für Industrielle Bauproduktion, Universität Karlsruhe, Germany

ABSTRACT: Computer-aided architectural design (CAAD) poses a number of novel challenges to database systems. The central thesis of this paper is that design decisions can be described in terms of constraints. Consequently one major concern of CAAD is the management of highly dynamic constraints. This paper motivates in detail the requirements for a constraint management component in building design, and presents a method for supporting the dynamic aspects of constraints.

1 INTRODUCTION

An architectural design process has many unique and individualistic traits: Its result is one–of–a–kind, the group of experts (architects, sanitary engineers etc.) differs from one building to the next, and each expert is only involved over part of the time. The cooperation among experts is more sequential than parallel, and, hence, becomes susceptible to misunderstandings or conflicts.

Similar shortcomings have been observed for the engineering activities in other disciplines as well, where they have in the meantime been overcome by computer–aided integration. Such an approach is possible in building design, too. The aim of "Integral Planning" ([14]) is to provide a much closer interaction between architects and other specialists without unduly restricting their creative work ([1]).

Creative work is a series of design decisions. This observation gives rise to two hypotheses underlying our work on computer support for Integrated Planning. For one, we tie design decisions to design areas encompassing a number of design objects rather than individual design objects. Second, we describe design decisions in terms of conditions that constrain the range of possible future design decisions.

Consequently, we base our support framework on the following concepts: A representation formalism for an 11-dimensional design space called the A4 space, a common database system where all design decisions can be stored as 11-dimensional data objects (so-called containers) and easily retrieved by spatial overlappings, and so-called constraints which represent design rules and design decisions and impose restrictions and interdependencies on the set of containers in the database.

2 CHARACTERISTICS OF CONSTRAINTS

Constraints are usually large in number and differ in type. Some reflect design decisions, others govern relationships between representations of the same real–world object, e.g., on different scales, further ones may have to do with general design rules or with design restrictions. The rules may be, e.g., organized within the following classes:

1. zoning and building regulations,
2. building norms,
3. expert–specific rules,
4. individual requirements of the building owner,
5. individual requirements of the architect.

Rules of the first two classes are of a more static nature. Those of the other three groups show a highly dynamic behavior. Expert–specific rules will be activated or deactivated several times during the design process, and the requirements of the building owner and of the architect reflect attitudes and may change over time.

This paper will present a method for supporting especially the dynamic aspects of constraints. Figure 1 shows a part of the design of the "cantonal school" of Solothurn, Switzerland, which we use to illustrate the characteristics of constraints. The figure shows the ground floor of the science wing of the school. All class–rooms on this floor (area A) are specialty rooms. Each room has been assigned to a particular teacher. Area B is a free communication zone for the students to meet during their breaks.

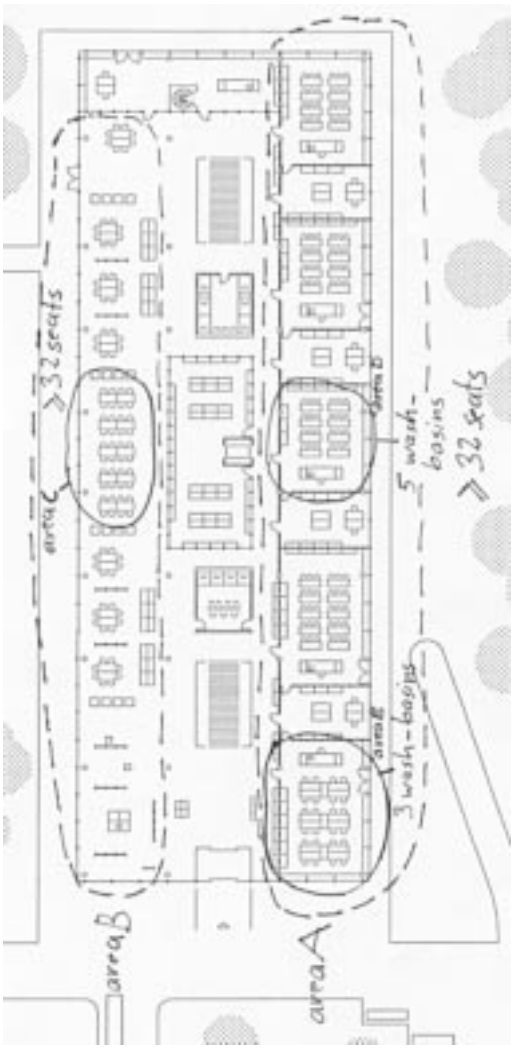

Figure 1: Cantonal school of Solothurn

Contrary to more traditional approaches, constraints are not bound to object types but to individual objects. E.g., for each specialty room in the example above different sets of constraints are valid: In one room the teacher needs three wash-basins, whereas in the other room the teacher needs five basins.

In fact, following our hypothesis it would be more consistent to bind constraints more generally to design areas. Indeed this makes good sense. Consider that the building owner may decide to have a minimum of 32 seats in some part of the free communication zone (area C), without determining the concrete area as yet.

A design process will result in a continuously increasing set of constraints, or more precisely, of active constraints which have to be controlled to ascertain that they are being satisfied. By contrast, passive constraints "lie in waiting". Consider an expert tool that reflects the work of a specialist, say a sanitary engineer. The tool will in all likelihood include a number of constraints to be satisfied once it gets to work. The constraints remain passive until then, and become active afterwards.

Typically, an expert tool is active only in a distinguished part of the building and corresponds to a special step within the design process. As a result, each area of the architectural design is described by a different set of constraints. Just consider that the ground floor may already be in the stage of planning the furniture whereas planning of the second floor is still on the functional level.

The set of constraints may vary even if no expert tool has been added. Because of new ideas and a deeper perception of the design problem, constraints may be dropped, new ones will be added, or the area in which the constraint is active may change.

The areas, in which constraints are active are defined not only by their geometric attributes, but also by other aspects, e.g. the current level of detail. A constraint which governs the arrangement of pillars is not yet relevant on a functional level of detail. Once the level of detail changes a large number of constraints must be included to obtain a consistent transformation.

Traditionally in databases, consistency is checked upon internally generated events, and prior changes are undone or must at least be corrected before the system allows one to proceed. By contrast, in architectural design, inconsistent states must be permitted to the extent desired by the user, checks should be initiated by user–defined, i.e., external events such as reaching a milestone, and responses to violations should be performed in a user–controlled fashion.

During a design step the constraints are analysed by a constraints checking facility. In support of "Integrale Planning", conflicts between different users should be detected as early as possible. The system should either notify the users, thus providing a platform for their cooperation, or where possible and meaningful should try to resolve the conflict, possibly with an explanation of the causes.

One aim of the project ArchE is to provide a constraint handling component in this highly dynamic environment. (Information about our project and the basic concepts can be found in [9, 12, 10, 11]).

3 TRIGGERING CONSTRAINT CHECKS

A standard technique to initiate constraints checking in databases at specifiable times is the concept of trigger ([2, 3, 6, 4, 5, 15]). Triggers normally consist of three parts: events, conditions describing violations, and actions, with the semantics that if an event occurs, the related condition is checked and, if true, the action is executed.

Our central idea is to augment triggers by a fourth component, **areas**, in order to account for our hypothesis and, hence, for the special design situations under which the constraint is applicable (or "active"). Elements of the area concept are all the attributes of the A4 space, with its dimensions of geometry (of the bounding box, where a condition is valid), several time attributes, resolution, size, type, alternative, and morphology. We call this kind of trigger an **area trigger**.

The execution model of area triggers is as follows. The part of the design on which the designer is operating determines the working area. During the time of operation, all triggers whose area overlaps with the working area, are taken into consideration. As such they are active, i.e., they become candidates for possible execution and hence, for constraint checking. Actual execution takes place on occurrence of the specified event within the working area.

Design decisions are reflected in changes of the attribute values of the area. A designer is also permitted to redefine the event that gives rise to the execution of a trigger, the condition under which the action will take place, or the action itself, thus adjusting constraints to a modified design status. A modified area trigger does not replace its earlier version but is instead appended to what is called the trigger history. Consequently, the entire time line of an area trigger remains available, an important feature that allows backtracking to earlier versions whenever one or more design decisions are revoked.

Cooperation between different designers takes place by the overlap of containers and areas. In particular, hence, design decisions and triggers, and thus constraints can be shared. In this way designers may immediately recognize potential conflicts if they attempt to associate an area with triggers contradicting each other.

The concepts of container ([12, 11]) and area trigger ([13]) provide a uniform framework to handle design decisions as constraints. This supports both, the control of expert-specific information and constraints, and the communication between different experts.

4 IMPLEMENTATION

The basic architecture of our CAAD environment is organized in two levels (figure 2). The upper level – the container model – is an immediate reflection of the users' perceptions of the design process and the design objects and, hence, has as its basic concepts containers for design areas and objects, and area triggers to reflect constraints as a combination of area of activity, events, condition, and actions. Basically, the container model is a translator between all views and actions at the user interface and corresponding internal data structures

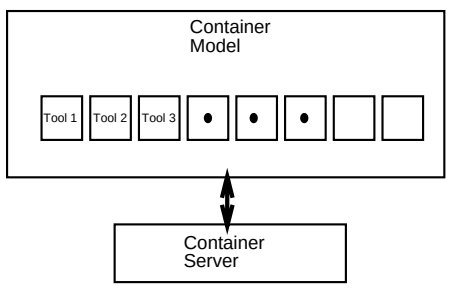

Figure 2: Architecture

and actions on them.

The lower level – the container server – is based on an object–oriented database system. Object–orientation was chosen because it is known to offer a particularly natural mapping from design objects (here: containers) and the actions defined on them to persistent storage.

The conceptual schema, i.e., the formal specification, for area triggers on the container server level is shown in figure 3. It consists of five parts which are combined into an AECAR schema. The schema can be instantiated to AECARs (Area-Event-Condition-Action-Rules). The area schema models the dimensions of the 11-dimensional design space and, additionally, some describing attributes necessary for trigger control. The AECAR-head integrates the events, conditions and actions which determine the activity in a specified area. Associated with the AECAR-head as a whole are further attributes which describe the overall state of an area trigger: *State of consistency* shows whether the constraint, in the specified area, forces consistency, permits inconsistency, or leaves consistency open. *Nullarea* indicates whether the area trigger has been predefined but is as yet not associated with any area. The attribute *user* specifies the expert tool which owns the area trigger. Each AECAR consists of exactly one area and one constraint condition, whereas several events and actions may be associated. Consequently, the validation of a constraint in a specific area may be initiated by different events, and several different actions may be triggered in case of a violation.

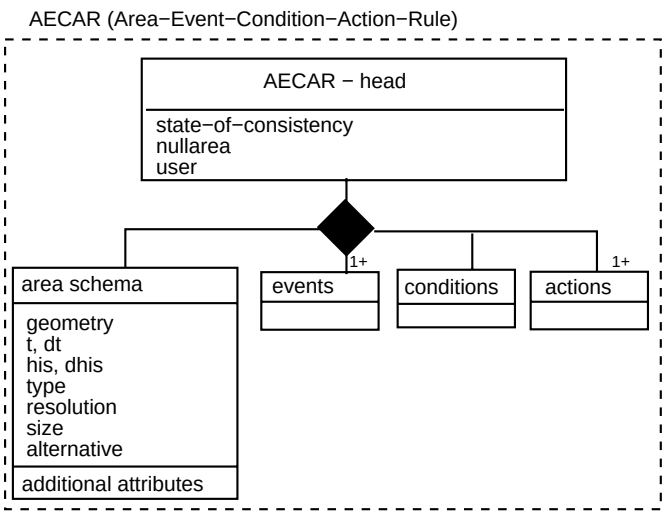

Figure 3: AECAR schema

5 DYNAMICS OF CONSTRAINTS

Each user of the design system will introduce his/her own constraints, relate them to a particular part of the design, and specify their behavior, i.e., the events initiating the validation of the constraint, and the actions to be taken in case of constraint violation. We expect these constraints to be frequently changed. For example, in figure 1 one of the teachers initially wishes to enforce 5 wash basins in his/her class–room. Therefore, the actions on a violation of that constraint should conform to a strict enforcement. Later on, the teacher may decide to loosen the requirement, so that the actions have to be changed in a way suitable to the new situation.

We now demonstrate in more general terms how the container model and the container server will deal with the dynamics of area triggers. Consider figure 4 as a schematic representation of the various (partially nested) design areas at the graphical user interface.

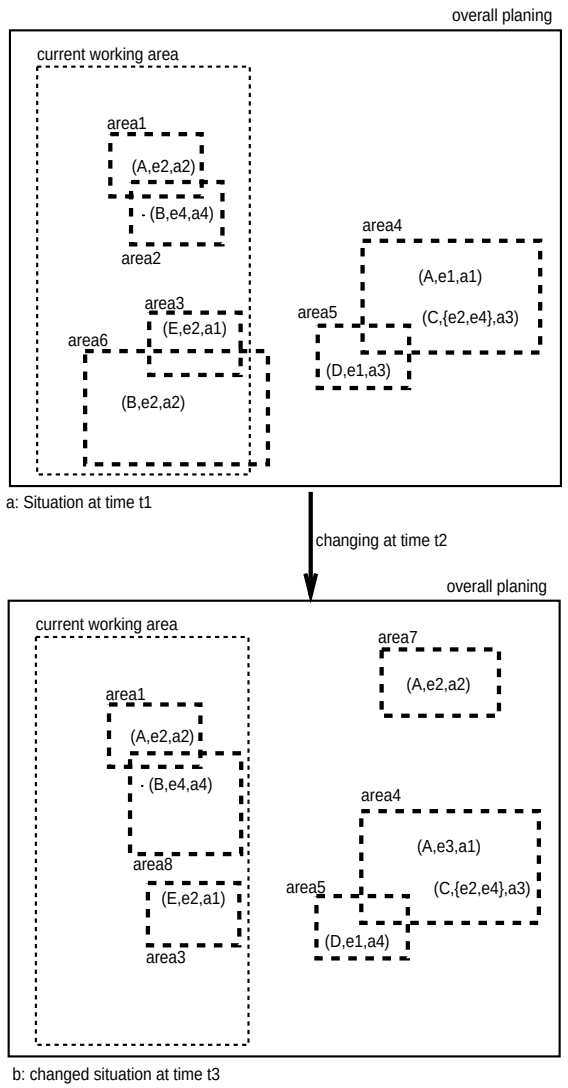

Figure 4: Dynamic changes of constraints during the design process

Figure 4a shows the consistency state of the design at time t1. The dotted boxes represent the areas in which the constraints are active. Each

area box contains its constraint – condition and behaviour, i.e., related events and actions. At time t2 the consistency constraints are modified resulting in the consistency state in 4b.

The example shows a number of typical changes:

- An existing consistency constraint becomes active in another area, e.g., at point t2 the constraint A is also activated in area7.
- The associated area of the constraint changes, e.g., area2 for constraint B is enlarged to area8.
- The events change as, e.g., for constraint A in area4.
- The associated actions change as, e.g., for constraint D in area5.
- A consistency constraint becomes inactive in a particular area, e.g., constraint B has been removed from area6.

Figure 5 shows at the container server level the corresponding AECARs for times t1 and t3.

Typically, a design process is not a sequential affair but one characterized by iterations and retractions. Consequently, our CAAD environment must allow arbitrary backtracking of the design for arbitrary parts. Consistency constraints must be able to follow these changes in design states. To do so, at each change of a constraint, i.e., the associated area or behavior, a new trigger object will be created. Therefore, with such an object we must associate the time period (kept in a history attribute) during which it is valid. During changeover, the endpoint of the history attribute of the old trigger object, and the starting point of the time interval of the new trigger object are set to the current time.

We note from figure 4 that the same constraint (more precisely: the same condition) may be valid in different parts of a design and show in each a different behaviour. Therefore, each consistency constraint in conjunction with the area, in which the constraint is active, needs a separate trigger object. Consider figure 4a. The consistency constraint A is active at time t1 in the areas area1 and area4. So, to describe this situation, two rules R1 and R5 are necessary (figure 5a). Similarly, the consistency constraint B is active in two areas, requiring two rules R3 and R6. The consistency

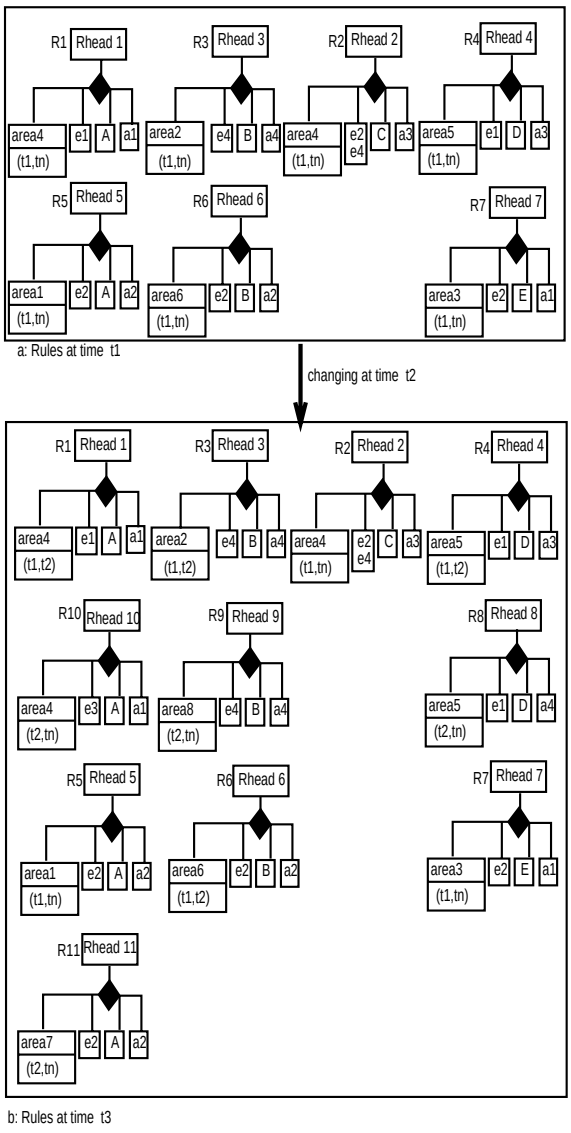

Figure 5: Dynamic adaptation of the constraints

check of the constraint C in the area4 may be enforced by two events reflected in figure 5a by a single rule R2. The consistency constraints D and E are, each, only active in one area. Correspondingly, there exist rules R4 and R7.

Figure 5 illustrates the development of the history attributes. Suppose that no rules existed in the database before time t1, i.e., that rule generation begins with time t1. Therefore, in figure 5a all of them contain the same starting point t1 in their history attribute. The end of the time interval is set to tn, which represents a time long away in the future, and means that the rule is currently active.

At time t2 all rules concerned have to be adapted to reflect the changes. The new set of rules is shown in figure 5b. For each change of an area rule the history attributes of the old and new rule must be set. We observe the following effects in our example:

- In area4 event e3, in place of e1, activates the checking of constraint A. Therefore, in rule R1 the history interval is closed, i.e., the end of the time interval is set to t2. To reflect the new situation, rule R10 is generated setting the history interval to start time t2 and end time tn.

- Constraint A is added to area7 where it is activated by event e2 and will cause, in case of violation, action a2. This is reflected in the new rule R11, with the history interval beginning at t2 and ending at tn.

- Constraint B in area6 is deactivated. Therefore, the history interval of rule R6 is closed by setting the end of the interval to t2.

- Constraint B is also active in area2, which is enlarged at time t2. For that reason, rule R3, in the same way as rule R6 above, is closed and a new rule R9 with the time interval beginning with t2 and ending with tn is created, corresponding to the new area8.

- In area5 the violation of constraint D is supposed to invoke action a3 instead of the former action a4. Accordingly, rule R4 is changed in the same way as rule R1 above, and rule R8 is created in a way similar to rule R10 above.

- The changes at time t2 do not affect the rules R2, R5, and R7. These are placed unchanged into the set of rules for time t3. In particular, their history intervals remain between times t1 and tn, indicating their continued actuality.

The architectural design does not restrict changes to certain fixed times. Openess to changes at any time is reflected in the possibility to set the time stamps to arbitrary values. As the example demonstrates, during the design process large sets of rules will evolve, that allow backtracking to arbitrary earlier time points in the design process. Since all rules are objects in their own right with no explicit connection among them, and any part of a rule may change on creation of a new rule so that one cannot derive an implicit connection either, backtracking must entirely rely on the time dimension and the related user–defined areas.

6 RULE EVALUATION

Typically, several persons work on the same design project in parallel. This will give rise to frequent conflicts among them. If we leave it to the persons to detect the conflicts these may go undetected for a long while, at least if the persons do not work simultaneously, or even forever, because the constraint complexity is not yet surveyable by persons. Consequently, conflict detection should be automated so that it takes place at the earliest possible time. Our rule evaluation mechanism is geared to do just that.

The foundation for conflict detection is the event. An event has a time dimension: it takes place whenever a designer performs a change to the design. An event also has a spatial dimension: it relates the change to a particular part within the design area. Constraint checking will now be invoked at the time of the event for all constraint areas which include the spatial dimension of the event. If there is more than one affected area, violation may point to a design conflict. If these areas are the responsibility of more than one person, cross–person conflicts become immediately visible.

Fig. 6 gives an example. The user triggers in his/her current working area the events E1, E2, and E3. CC1, CC2 und CC3 are examples of areas of rules. CC1 is completely located within the actual working area, whereas the area of CC2 only overlaps with the actual working area, and the area of CC3 falls completely outside of it. Because the user is only able to trigger events within his/her

current working environment, only rules CC1 and CC2 are candidates for violation. The spatial dimension of event E1 is completely outside the areas related to CC1 and CC2. Therefore, for the consistency check triggered by event E1 neither CC1 nor CC2 are relevant. E2 lies within the area of CC2, i.e., the effect of E2 in the design overlaps with the position of CC2. In this case CC2 has to be proven valid but not CC1. Suppose now that a second designer operates in a working area that is disjoint from the working area shown, but also overlaps with CC2. Then a violation within CC2 may affect him/her as well. This is not the case for E3 and CC1 which are restricted to the user of the working area shown.

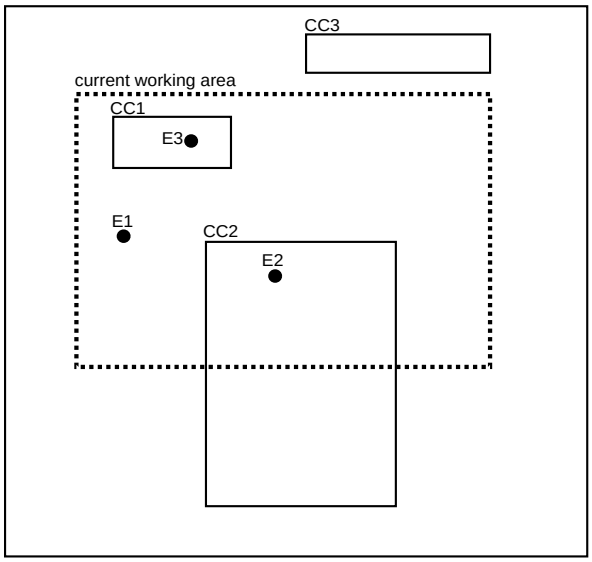

Figure 6: Relationship between events and constraint areas

The execution model must facilitate the checks as discussed above. Since the working areas of designers remain fairly stable over some time, some efficiency can be gained by precomputation. On the other hand, because designers may work in parallel, they may generate events concurrently so that the execution model must be able to deal with sets of events.

Suppose that all rule areas have been precomputed, which overlap with the current working area (we refer to these rules as O–rules). Given a specific working area, checking is done in the following steps:

1. Select all O-rules.
2. From the set of O–rules select those for whose related areas events have been raised.
3. Test for these rules whether an event applies.
4. Where it applies, check the related constraint.
5. If the constraint is violated, take the corresponding action.

For the selection of the O–rules, overlapping is defined by the following conditions that must simultaneously hold:

- The geometrical areas overlap.
- The current system time is included in the history interval of the rule.
- The values of the discrete rule dimensions equal the corresponding variables of the current working area.

As far as the actions are concerned, notification of the designers via the user interface may in many cases be sufficient. Where actions take compensatary steps the database state may change, necessitating a new check of the O–rules.

Computation of the set of O–rules can be made more efficient by the use of a multi–dimensional access path. These access paths provide fast access to the areas overlapping with the selection criterion. From the selected areas the constraints of interest, with their events and actions, may very easily be found on the basis of the structure of the AECARs. The technique applies to a current working area as a selection criterion as well as to any arbitrarily defined area in which the user may show an interest. We provide a single access path covering all AECARs regardless of which user defined it.

7 CONCLUSION

We claim that area triggers are a novel and highly promising concept to describe the progressively more restrictive design space in which a designer operates as the design evolves. We also claim that area triggers maintain the flexibility of creative design, by taking the current situation into account, giving the designer latitude to determine the degree of consistency desired, and permit him/her to retrace or revoke earlier design decisions. On the other hand, to the extent that constraints have been defined and activated, he/she obtains automatic support for enforcing design decisions taken by him/her or - through the overlap of areas - by other experts.

We are currently validating the area trigger concept on the basis of an object-oriented implementation of containers, triggers, the underlying database and a graphical interface, using two design expert tools, the component-based steel system MIDI ([7]) and the installation planning tool Armilla ([8]).

References

[1] K. Abramowicz, B. Boss, V. Hovestadt, J.A. Mülle, R. Sturm, and P.C. Lockemann. Konsistenzüberwachung in Datenbanksystemen - Eine Anforderungsanalyse anhand der Entwurfsbereiche Architektur und Schiffbau. In *Datenbanken in Büro, Technik und Wissenschaft (BTW)*. Springer Verlag, Reihe Informatik Aktuell, 1995.

[2] P. Bayer. State-of-the-art report on reactive processing in databases and artificial intelligence. In *The Knowledge Engineering Review, Vol 8:2*, pages 147–171. Cambridge University Press, 1993.

[3] S. Chakravarthy, V. Krishnaprasad, Z. Tamizuddin, and R. H. Badani. ECA Rule Integration into an OODBMS: Architecture and Implementation. Technical Report UF-CIS-TR-94-023, University of Florida, May 1994.

[4] O. Diaz, N.W. Paton, and P. Gray. Rule Management in Object-Oriented Databases: A Uniform Approach. In *Proc. of the 17th Int. Conf. on Very Large Data Bases*, pages 317–326, Barcelona, Spain, 1991.

[5] S. Gatziu and K.R. Dittrich. SAMOS: An Active Object-Oriented Database System. *IEEE Quarterly Bulletin on Data Engineering*, 15(1-4):23–26, December 1992.

[6] N. H. Gehani and H. V. Jagadish. Ode as an Active Database: Constraints and Triggers. In *Proc. of the 17th Int. Conf. on Very Large Data Bases*, pages 327–336, 1991.

[7] F. Haller. midi 1000, tragwerk planungsgrundlagen. Technical report, USM bausysteme Haller.

[8] F. Haller. *Armilla – ein Installationsmodell.* Institut für Industrielle Bauproduktion, Universität Karlsruhe, 1985.

[9] F. Haller, K. Friedrichs, V. Hovestadt, P. C. Lockemann, J. A. Mülle, and R. Sturm. The Design Navigator. In *Proc. 5th Int. Conf. Computing in Civil and Building Engineering, Anaheim, California*, pages 335–340. Computing of V-ICCCBE, 1993.

[10] L. Hovestadt. *A4 - Digitales Bauen - Ein Modell für die weitgehende Computerunterstützung von Entwurf, Konstruktion und Betrieb von Gebäuden.* Fortschrittsberichte VDI, Reihe 20 Rechnerunterstützte Verfahren Bd. 120, Düsseldorf, Dissertation, Institut für Industrielle Bauproduktion, Universität Karlsruhe, 1994.

[11] L. Hovestadt, V. Hovestadt, J. A. Mülle, and R. Sturm. ArchE – Entwicklung einer datenbankunterstützten Architektur - Entwurfsumgebung. Technical Report Nr.23/94, Universität Karlsruhe, November 1994.

[12] P. C. Lockemann, J. A. Mülle, R. Sturm, and V. Hovestadt. Modeling and integrating design data from experts in a CAAD-environment. In *Proc. of. the European Conference on Product and Process Modelling in the Building Industry, 1994*, to appear.

[13] R. Sturm and P.C. Lockemann. Bereichsdynamische Konsistenzüberwachung. Technical Report 24, Universität Karlsruhe, 1994.

[14] P. Suter, R. Gfeller, N. Kohler, and J. van Glist. Haustechnik in der Integralen Planung. Bundesamt für Konjunkturfragen Bern, 1986.

[15] J. Widom. The Starburst Active Database Rule System. *IEEE Transactions on Knowledge and Data Engineering*, to appear.