Temporized and localized rule sets

Rose Sturm, Jutta A. Mülle, Peter C. Lockemann

Institut für Programmstrukturen und Datenorganisation
Fakultät für Informatik
Universität Karlsruhe, 76128 Karlsruhe
[lockeman|muelle|sturm]@ira.uka.de

Abstract. Constraint management plays an important role in design applications where constraints reflect design restrictions and design decisions. ECA rules are a widely used mechanism to enforce constraints. The paper argues that such rules must be augmented for design environments by a spatial and a temporal dimension of validity, resulting in so-called area-event-condition-action (AECA) rules. The spatial dimension allows to restrict constraints locally in the design space, and to control interaction between designers. The temporal dimension permits designers to retract their designs to earlier stages.
The paper introduces the concept of AECA rules, motivates them by examples from building design, discusses rule management, and then introduces two important issues, conflict detection during collaboration, and backtracking during design revision.

1 Introduction

Active rules (also variously called database triggers, event-condition-action rules, or ECA rules) are perceived, by researchers and database system (DBS) vendors alike, as an ideal means to structure both the static and dynamic properties of complex applications. In an – otherwise critical – paper Simon and Kotz [12] list as two of their benefits a clear separation from and explication of business rules from application programs with better transparency to and control by the business organization, and performance improvement by restricting or concentrating the auditing of applications to well-defined and controllable events. Rules blend in naturally with constraints because constraints have traditionally been used to factor out application semantics common to a number of application programs from these programs.

Since database schemas have over decades been another means for factoring out common semantics, rules have usually become part of a (perhaps extended) database schema. However, such an approach imposes severe penalties: it ties the rules to object types (with "object" being used in a general sense), that is to a class of similar instances, and it also ties them to the entire life time of the type definitions. In other words, it imposes given rules uniformly over space (all instances) and time (the entire life time) of a type.

This work was in part supported by the German Research Council (Deutsche Forschungsgemeinschaft) under contract no. Lo296/11-1

On the other hand, there are applications where it is most natural for the users to confine the validity of constraints in space, i.e., to subsets of instances of one or more types, and the enforcement of the corresponding rules in time, i.e., only to certain periods of time. Besides, technically such a limitation would permit a more selective and, consequently, more efficient constraint evaluation and enforcement.

Restricting rules in space and time depends on the notion of "space" and "time" used. Different applications may use slightly different notions. It makes sense, therefore, to take a top–down approach in order to develop, or select, the appropriate mechanisms for rule management and enforcement. For this paper we choose an application scenario taken from building design. Implementation issues are only briefly touched upon although we are aware that the added expressiveness of these concepts imposes new challenges that may negate some of the technical benefits associated with the restrictions.

The paper ist structured as follows. In chapter 2 we develop the user's perception and needs for highly dynamic rule management. Chapter 3 relates the technical requirements to rule management to previous work. Chapter 4 discusses our basic approach to deal with the localized and dynamic aspects of rules by introducing area-event-condition-action rules and a concomitant execution model which allows to deal with an essential consequence of design activities, conflict detection during collaboration. Chapters 5 and 6 demonstrate the combined effects of spatial and temporal aspects for rule management, with chapter 6 demonstrating a second consequence of design activities, backtracking during design revision. Chapter 7 outlines the system implementation, and Chapter 8 concludes the paper.

2 A design application scenario and its requirements

In applications like building design, the result of the design process is "one-of-a-kind". Common to all such design processes is the dearth of useable earlier results and, hence, the vague, and poorly structured data at the beginning, that become more and more concrete over the course of the process. Initially, the designer's freedom of action is only circumscribed by general design rules and physical laws, but as the design progresses more and more rules are added. Also, design decisions are often revised, and alternatives pursued. Different parts of the design exist in different states of detail at the same time.

The premise of this paper is that design decisions can be considered as constraints on the degrees of freedom that a designer may have. Consequently, the definition and enforcement of constraints and rules must follow the decision process. Constraints develop over time, and they may be retracted or revised, invalidating or modifying some constraints or reverting to earlier constraints. Similarly, constraints evolve differently in different design spaces and, hence, should be tied to particular design spaces.

Design processes are "many–person". To meet short design times under guaranteed quality, the steps of such a process and the experts contributing to them

must be closely coordinated, particularly in light of the increasing interleaving of their activities with their tight feedback loops in place of the more traditional sequential processes. Since designers work on different parts and to different degrees of detail, the set of shared constraints is again a subset of all constraints and may vary over time and space.

Let us briefly illustrate these aspects. Figure 1 shows a part of the design of the ground floor of the science wing of a school in Switzerland. 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.

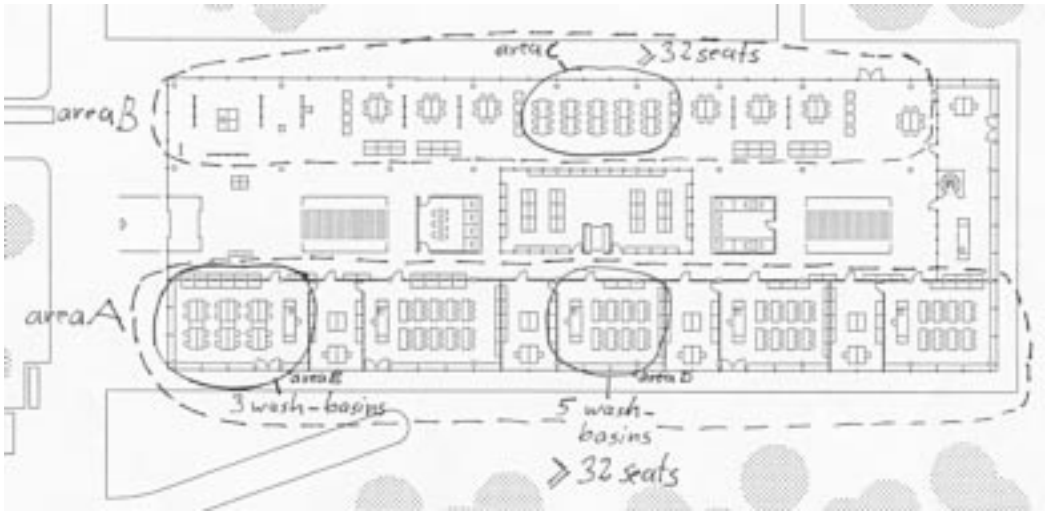

Fig. 1. Design for the Cantonal school of Solothurn

The ultimate goal of the designer is the design of objects with well-defined physical or conceptual boundaries. Examples of these so-called *design objects* are classrooms, seats, exhibition stalls, plumbing and wiring. Design objects may be contained in others, such as seats in a classroom, or may overlap with others, such as plumbing and classrooms. Obviously, then, there is a multitude of relationships between objects, and the designer deals with them by imposing a spatial structure, a *design space*, on them which circumscribes the objects whose interrelationships are of current interest to him. Take as an example a set of classrooms together with that part of the wiring and plumbing that directly affect them, in order to study the effects of a change of the number of water and electricity outlets in a classroom.

Constraints may be bound to design object types and also to individual objects. Specialty rooms may require, in general, water and electricity outlets. Their particular number may, however, be an invariant of an individual room.

For example, it may depend on the number of wash basins desired by each teacher. On the other hand, where an object does not yet exist, constraints can only be associated with design spaces. Consider that the building owner may decide to have a minimum of 32 seats (area C) in some part of area B, without determining the concrete position as yet. Finally, even where all objects are known, a constraint may be more naturally be associated with a design space. Take a geometric area encompassing part of the classroom area and the public area where only a certain maximum number of outlets is permitted in total. Take as another example different levels of detail in planning where 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. Since design objects have themselves a certain spatial extent, the unifying notion for circumscribing the range of constraints is that of design space (or design area).

Constraints evolve over time. For example, the constraint for area C may only be introduced after the size of the classrooms has been determined, and may later be changed again. Constraints may also be "canned". 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 "lie in waiting" until then, and become active afterwards, i.e. must be enforced from then on.

The architectural design process is an iterative procedure. Very often, particular parts of the design will be set back to an earlier state, and will then be the new starting point for further design. This feature will cause a backtracking of all relevant constraints in that part of design. For example, suppose new information about the number of future pupils becomes known, causing the need for two more specialty rooms. Therefore area B must be revisited and set back to an earlier state. This area-oriented backtracking concerns physical design decisions as well as constraints related to them, and may affect other areas or objects such as those of the sanitary engineer.

Enforcement of constraints is uneven. A point in time where checking is mandatory in order for the design to proceed is referred to as a milestone. In between milestones, however, only certain consistencies – in particular those that are time–invariant such as zoning and building regulations, building norms, or physcial laws – should continuously be evaluated whereas other inconsistencies regarding, e.g., individual requirements of the building owner or the architect should be tolerated, and not enforced automatically. For example, if a wash basin has been placed one should immediately check if a sewage pipe with enough sloping can be installed. If not, the reaction must also be immediate: Either the wash basin must be placed at a different location, or extensive parts of the existing plan for sewage pipes have to be changed. On the other hand, a constraint such as to have a minimum of 32 seats in the free communication zone, may vary in its importance during the design.

Reactions to a violation even of the same constraint may vary over time and space. For example, a constraint may have low priority at an early stage of the design but may become essential later on, such as the position of the

plumbing outlets. A change of the number of wash basins may be prohibited in one classroom but open to negotiation in another.

3 Related work

A standard technique to deal with constraints, with the flexible initiation of constraint checks, and the specification of appropriate responses to violations are ECA rules; these are considered as suitable means to support the above described requirements. An ECA rule is triggered into execution on some event such as a milestone, a vital design operation or a designer's spontaneous wish for a consistency check, causing it to evaluate the associated constraint condition and in case the condition holds, i.e. the constraint is violated, to perform the associated action.

The corresponding concepts, languages and computation models have been widely explored. In most approaches, though, rules are either ubiquitous or tied to object types. For example, ADAM/EXACT [5] indexes rules by class: the class-rules attribute has as its value the set of rules to be verified when a message is sent to any instance of this class. More flexibility can be found in the relational systems Starburst ([14]) and Postgres ([13]), both of which allow to organize rules in rule sets. However, each rule must explicitly be placed in a rule set by the user. This is definitely a too low-level mechanism for our purposes, since we are in a position to attach to rule sets a more specific semantics of areas. In particular, it does not allow to exploit the cross-effects between rule sets arising from the overlap of areas

Flexibility in event handling has recently attracted considerable attention, particularly in the context of object-oriented, active database systems. SAMOS ([7]), Sentinel ([3], and Ode ([9] support simple and complex events, that are constructed with event algebras. REACH [1] also introduces the notion to milestones that are to be detected. SAMOS even takes much broader contexts int account by supporting "event parameters" that may be used to restrict complex events to specific relevant categories, e.g., only to the events within one transaction, or to all events raised by a particular user. Similarly, Sentinel [4] provides four different rule evaluation contexts which differ in the set of relevant events which are used to detect complex events. Some systems allow to relate time stamps to events with the meaning that the rule will fire only if the event arises in the defined time period.

It appears, then, that the modern aproaches to event management are flexible enough to deal with the spatial aspects of our environment. We submit, though, that to do so would overload the event mechanism and blur the distinction between the spatial effects with our special area semantics and the temporal effects.

Finally to the best of our knowledge, temporal validity (or lifetime) of rules has not explicitly been dealt with in the literature (and implicitly only within the context of time stamping of events) and, hence, backtracking of rules has so far not been an issue.

4 AECA rules

Our premise is a clear separation of spatial, temporal and lifetime aspects in rule management. Consequently, we propose to extend ECA rules by a fourth component that explicitly accounts for the spatial aspects, and a rule header to account for the lifetime aspects. The result is what we call *area-event-condition-action (AECA) rules* (or for short: AECARs).

The structure of AECARs is illustrated in the form of a conceptual schema in figure 2. The basic premise for organizing the rules is that each constraint is represented, in the context of an area, by a single rule. Consequently, each AECAR consists of exactly one area and one constraint condition, whereas several events and actions may be associated, so that 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. Clearly, a constraint may appear in as many rules as there are areas which it presently is associated with. Likewise, an area may appear in as many rules as there presently are constraints associated with it. The validity (time) interval for the rule is, through the rule head, associated with the rule as a whole. As we shall see below, this allows to establish a history of design decisions and to permit backtracking of decisions to a specific point in time. The overall structure permits us to leave the condition (and thus, constraint) part and the action part unimpeded by any temporal and spatial considerations.

The application scenario seems to suggest that a design area is a purely two-dimensional geometric affair. However, our partners from the school of architecture define areas in a more far-reaching sense. Areas are placed within a seven-dimensional design space that formalizes all design decisions from all design phases according to the geometrical coordinates, time, resolution, size, type, user, and morphology, etc. in a uniform manner. These attributes are called **dimensions** of the multi-dimensional design space. Within this design space each design object or area represents a hypercube or, if all dimensions are instantiated, a point. (Information about our project and the basic concepts can be found in [10, 11]).

Associated with the AECAR head are attributes which describe the overall state of an area rule: *State of consistency* shows whether the constraint, in the specified area, forces consistency, permits inconsistency, or leaves consistency open. *History* holds the validity interval of the rule. The other two areas have to do with "canned" rules which as part of expert tools come into play only after the design reaches a certain stage. *Nullarea* indicates whether the rule has been predefined but is as yet not associated with any area. The attribute *user* specifies the expert tool owning the rule.

Area definitions, event specifications, constraints, and action specifications are declared individually and are then combined on a case-by-case basis via a rule header into a rule either statically as part of a general framework or expert tool or dynamically as the result of interactive design decisions. In particular, this allows the placement in a library and the reuse of any of them.

Consider as an example rule the application scenario of figure 1. The user

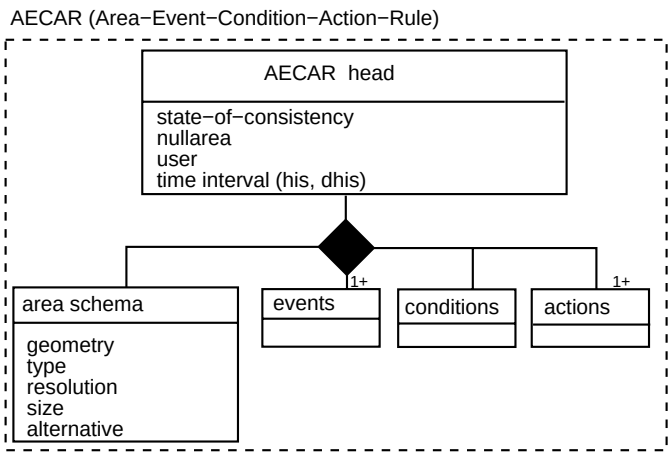

Fig. 2. AECAR schema

may interactively delineate and declare area E together with an event, resulting in the declaration (informally stated)

Area: geometry: (x,y,z)-coordinates of area E in figure 1
type: {functional planning, installation planning}
resolution: *,
size: *,
alternative: *.
Event: delete (wash-basin),

At some earlier time, he/she may already have entered into the library the desired constraint condition and action ("notify" refers to a programmed procedure).

Constraint condition: number of wash-basins ≥ 3,
Actions: notify (installation engineer), notify (user).

The four are combined into a rule by the user with the attributes of the AECAR-head set by the system (for the setting of *History* see chapter 5).

The rough execution model is as follows. During his/her work a designer designates a working area on which he/she is operating. Note that this area need not be identical to the area of any AECAR or design object. Hence, during the time of operation, all rules whose area overlaps with the working area must be 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. This requires an extended notion of event which combinesspatial and temporal aspects into a pair (point in time, area of occurrence). We refer to this kind of event as an area event.

Fig. 3 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 current working area, whereas the area of CC2 only overlaps with the current 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.

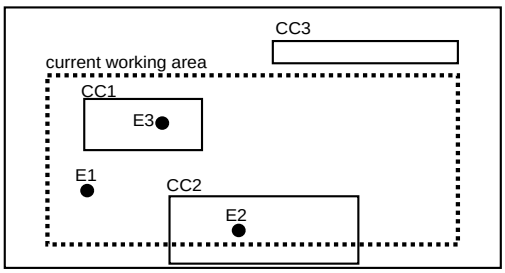

Fig. 3. Relationship between events and constraint areas

AECARs are well-suited to deal with collaborative behavior between several designers. Usually, such collaboration will give rise to conflicts. If we leave it to the persons to detect the conflicts these may go undetected for a long while. Now, if rule execution affects more than one working area, the violation may point to a design conflict. If these areas are the responsibility of more than one person, cross-person conflicts become immediately visible. Take the previous example and suppose 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.

In more detail, the execution model looks as follows. Given a specific working area, checking is done in the following steps:

1. Select all rules which overlap with the current working area.
2. From the set of these rules select those for whose related areas events have been raised.
3. Test for these rules whether a given event applies.
4. Where it applies, check the related constraint.
5. If the constraint is violated, take the corresponding action.

Overlapping is defined as an overlap of the geometrical areas together with the identity of the values of the discrete dimensions of the rule area to those of the current working area, provided the current system time is included in

the history interval of the rule. As far as the actions are concerned, notification of the designers via the user interface may in many cases be sufficient. Where actions take compensatory steps the database state may change, necessitating a new check of the rules. Note also that 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.

5 Rule management: Spatial effects

Design decisions will often affect the rules. Consequently, a designer is permitted to redefine the area of a rule, the event that gives rise to the execution of a rule, the condition under which the action will take place, or the action itself, thus adjusting constraints to a modified design status. Take as an example that the designer no longer tolerates that the issue of placing the 32-seat zone remains unresolved. Hence, the area becomes more narrowly circumscribed, events for checking the constraints become more frequent, more constraints must be observed, and violations must immediately be compensated for.

A new design decision results in a new AECAR with a new validity interval whereas the AECAR it replaces remains in the system with a now closed interval. This, incidentally, is another argument for the decomposition of AECARs into independent components.

We illustrate the general principle. Consider the left side of figure 4 as a schematic representation of the various (partially nested) design areas on the graphical user interface.

Figure 4–A1 shows the users' views of the consistencies at design at time t1. The dotted boxes represent the working areas of several users. For each area the figure notes its constraint, related events, and actions. At time t2 the consistency constraints are modified resulting in the view of 4-A2.

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 form a new area 8.
- 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 4-B shows the corresponding AECARs for times t1 and t3. Compare figures 4-A1 and 4-B1. The consistency constraint A is active at time t1 in two areas, area1 and area4, though with different behaviour. Hence, two rules R1 and R5 are necessary. Similarly, the consistency constraint B is active in two areas, requiring two rules R3 and R6. On the other hand, a single rule R2 indicates that the consistency check of constraint C in area4 may be enforced by two

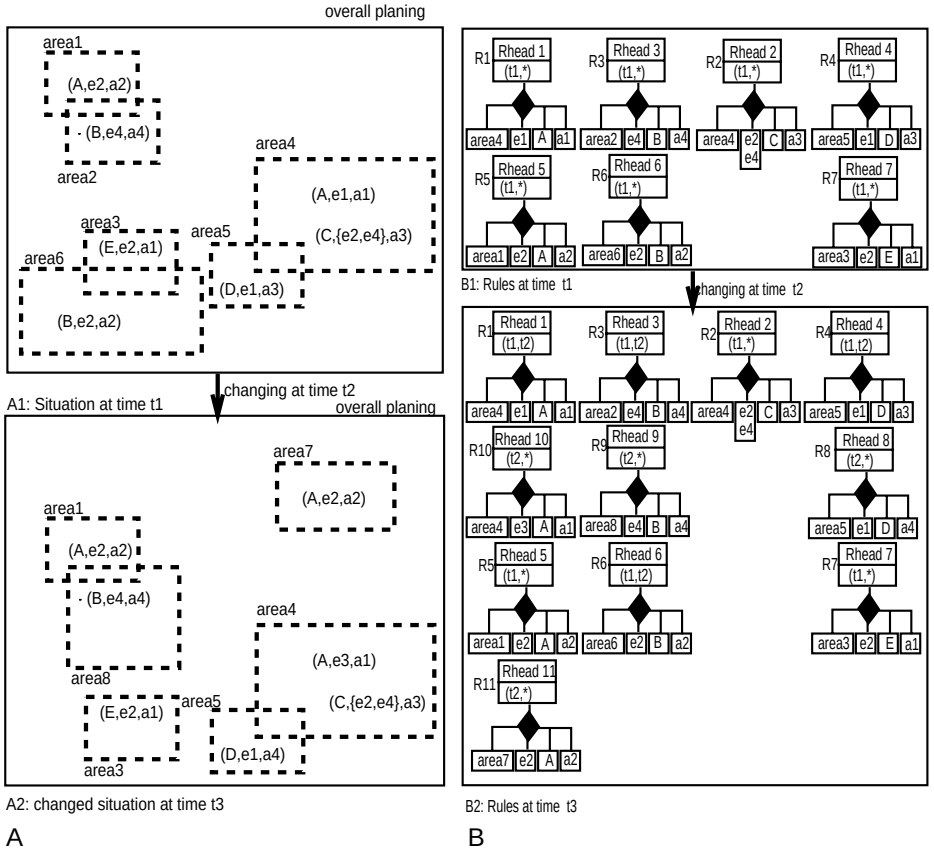

Fig. 4. Dynamic changes of constraints during the design process

events. The consistency constraints D and E are, each, only active in one area. Correspondingly, there exist rules R4 and R7.

Figure 4-B also illustrates the development of the history attributes. Whenever a new rule is created, the endpoint of the history attribute of the old rule it supersedes, and the starting point of the time interval of the new rule are set to the current time. Suppose that rule generation begins with time t1. Therefore, in figure 4-B1 all rules contain the same starting point t1 in their history attribute. The end of the time interval is set to *, which represents a time long way into the future.

At time t2 all rules concerned have to be adapted to reflect the changes. The new set of rules is shown in figure 4-B2. We observe the following effects:

- 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 *.

- 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 *.
- 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 * 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 *, indicating their continued actuality.

6 Rule management: Temporal effects

As the example demonstrates, the rule base preserves the entire history of design decisions. As an important consequence, the approach introduces the capability to backtrack the design of arbitrary user-defined parts. Very roughly speaking, let a designer specify an arbitrary backtracking area BA, and a setback time t. Within BA the situation is restored to the one that existed at t, that is, all rule areas within BA are recovered as they existed at t, and all decisions taken after t are simply purged from the database.

In fact, there is a complicating factor in backtracking in that the architect may delineate an arbitrary area as a backtracking area. In other words, the backtracking area may not bear a causal relationship to either the current design areas or the earlier ones. Consider figure 5 which continues figure 4 to a point t5 in time. At time t4 the user identifies a backtracking area as shown in the center left. Apparently, the backtracking procedure must somehow involve all areas overlapping with the backtracking area. The question, then, is whether backtracking should affect these areas in their entirety, even those parts that lie outside the backtracking area.

Suppose now that the backtracking area is (at current time t4) to be set back to time t1. In a first step the rules in the current database (B2 in figure 5) with areas overlapping the backtracking area must be identified. In the example there are six such rule areas in the database B2: area3 (with rule R7), area4 (with rules R1, R2 and R10), area5 (with rule R8), area6 (with rule R6), area7 (with rule R11), and area8 (with rule R9). In a second step the status of the constraints of these rules at the time to which the design situation is to be restored (setback time) must be determined.

We illustrate the restore actions with our example.

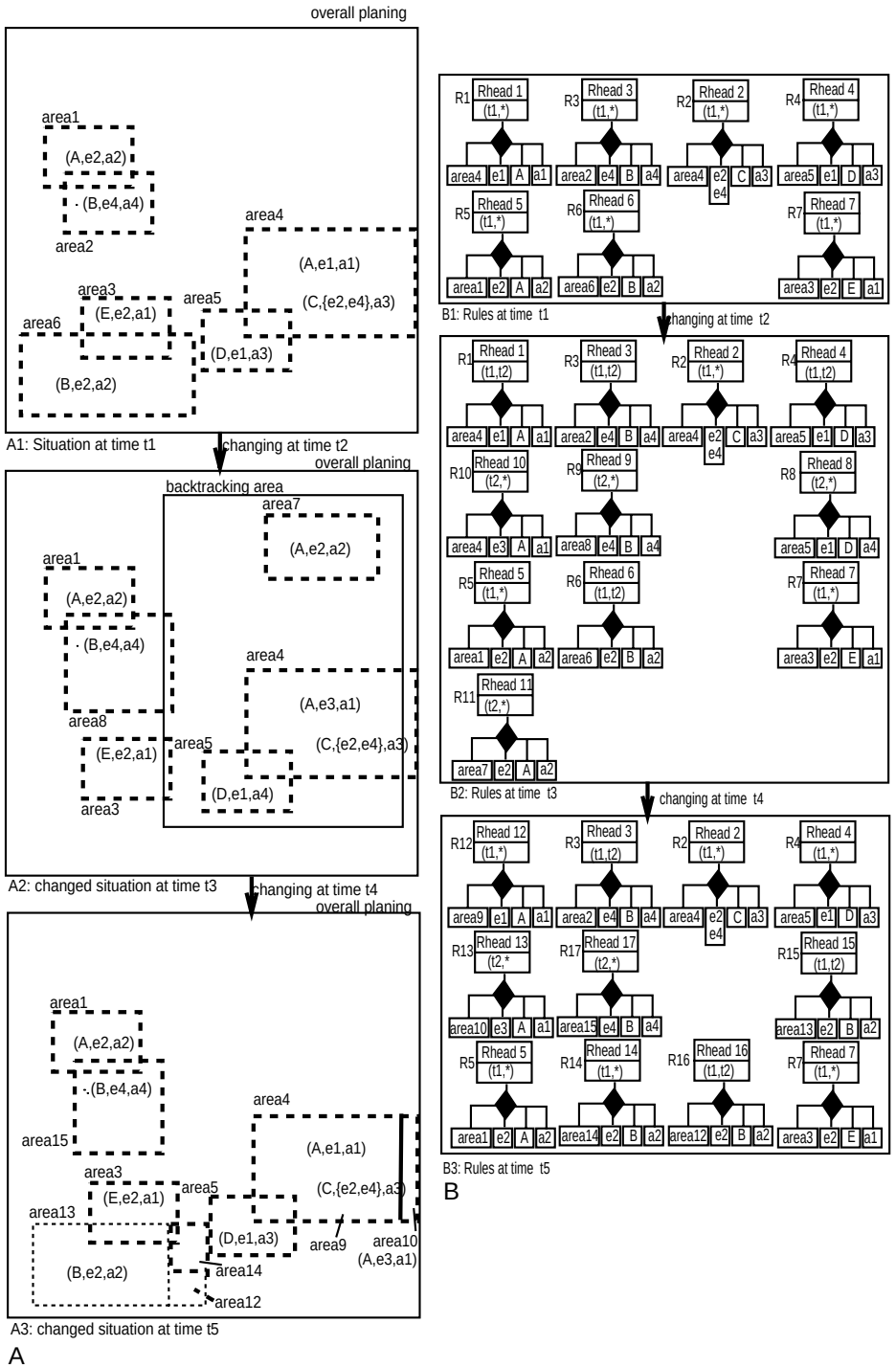

Fig. 5. Dynamically backtracking of parts of the design

- Rule R5 with area1 lies completely outside the backtracking area and, hence, does not change.
- Rule R8 with area5 lies completely within the backtracking area but its validity interval does not include t1 because at t2 the action was changed from a3 to a4. Therefore Rule R8 is deleted. The validity interval of rule R4 with area5 includes t1. Because interval endtime t2 extends into the future as seen from the perspective of t1, R4 is reactivated with the validity interval (t1, *).
- Rule R11 with area 7 lies completely within the backtracking area and was generated at time t2. Because rule R11 was not active at t1, R11 is deleted.
- Rule R7 is not affected although its area3 overlaps with the backtracking area, because its validity interval includes t1.
- The same is true for rule R2 and area4.

This leaves area4 with rule R10 and area8 with rule R9, which overlap with the backtracking area. In order to deal with them we distinguish between *closed* constraints which if valid in an area are valid in each subarea, and *open* constraints, otherwise. For example, physical laws are closed constraints whereas design decisions such as the 32-seat requirement usually are not. We can then establish as a procedure that for closed constraints only the subarea within the backtracking area is rolled back whereas for open constraints the entire area is rolled back. For closed constraints this entails splitting the area. For the pieces inside the backtracking area, the old state must be regenerated whereas outside the current state must be preserved. Note that the change of an area results in the generation of a new rule so that for each split part a new rule must be established. Now suppose that constraints A and B are closed. Accordingly:

- Rule R10 with area4 gives rise to new rules by splitting area4 into two parts. Outside the backtracking area (area10) the same composition of events, condition, actions as in R10 is valid. This is postulated as rule R13 with the area10. Inside the backtracking area (area9) the status at t1 must be regenerated. R12 is the updated rule R1 with the new area area9 and the new validity interval t1 to *. R10 is then deleted.
- For rule R9 area8 must be similarly split. The subarea within the backtracking area would have to be restored to its state at time t1. No corresponding rule can be found inside this area. This leaves R9 adjusted for the area outside the backtracking area area15, which is postulated as rule R17. R9 must be deleted.

Finally, there is the not-so-obvious case of rule R6 which with area6 does not exist any longer at time t4 but existed at restoration time t1 and overlaps with the backtracking area. Its constraint B is closed so that the splitting principle applies again. Rule R6 must be restored for the subarea area14 which lies inside the backtracking area. This is described by rule R14 which is identical to R6 except for the area14. To generate the correct situation outside of the backtracking area, two new (rectangular) areas area13 and area12 must be established. Correspondingly, two new rules R15 and R16 are created, which except for the

areas are copies of R6 and hence have a time interval (t1,t2), so that they are not active at time t5.

7 System environment

Rule management as discussed in this paper is part of a three-layered implementation architecture for a CAAD environment. The topmost layer – the representation layer – reflects the users' perceptions of the design process and the design objects and areas. An elaborate user interface allows the architect to deal not only with the geometrical confines of areas but also with the other dimensions ([10]). Specifically for the purposes of this paper, he/she may define and associate rules with areas as a combination of area of activity, events, constraints, and actions, and he/she may delineate working areas and raise events within them. Through the overlap of areas the interaction between different experts during the design becomes immediately visible to them in the form of additional sets of constraints to be observed as the result of a design decision in some other areas.

On the next lower layer all concepts on the user interface are represented in a uniform (conceptual) data structure, the so-called container. For the purposes of this paper, the most important functionality of the layer is the management of the rule base according to sections 5 and 6 as well as provision of the execution machine for the rules according to section 4, essentially as a trigger mechanism.

On the lowest level – the container server – the containers are implemented via an object-oriented database system. This layer also directly utilizes performance enhancing measures such as the precomputation of the set of overlapping rules for a given working area and multi-dimensional access paths to containers and to AECARs to allow arbitrary selection criteria for the rapid identification of relevant AECARs according to our in section 4 introduced event notion.

8 Conclusion

Our paper deals with constraint management. Taking a purely top-down view we have argued that at least in certain design environments ECA-rules have a strong flavor of spatial and temporal validity. We suggested a more discriminating mechanism of area-event-condition-action (AECA) rules. They allow to impose constraints locally in the design space, to control interaction between designers, and to permit designers to retract their designs to earlier stages. We also demonstrated that the combination of these dimensions during backtracking may give rise to complicated situations.

By building AECA rules around the real needs of building designers and their view of the design process and design decisions, we are in the fortunate situation of having a testbed and test persons to evaluate our approach from both a technical and an applications standpoint. We developed an extensive set of constraints for an expert tool, and are currently implementing the testbed.

Much remains to be done. Foremost are experiments with our collaborators from the school of architecture whether AECA rules offer the necessary flexibility without imposing undue technical penalties. If successful, much more attention must be paid to the issue of designing constraints and rules, if possible even spontaneously. Third, while our current testbed implementation does not consider efficiency as an overriding issue, the ultimate success will depend on good performance of the AECA rule mechanism.

References

1. A.P. Buchmann and H. Branding and T. Kudraß and J. Zimmermann. Rules in an Open System: The REACH Rule System. In N.W. Paton and M.H. Williams, editors, *Rules in Database Systems.* 1st Int. Workshop on Rules in Database Systems, Springer Verlag, 1994.
2. S. Chakravarthy, B. Blaustein, A.P. Buchmann, M. Carey, U. Dayal, D. Goldhirsch, M. Hsu, R. Jauhari, R. Ladin, M. Livny, D. McCarthy, R. McKee, and A. Rosenthal. HIPAC: A research project in active, time-constrained database management. Technical Report XAIT-89-02, Xerox Advanced Information Technology, Cambrige, Massachusetts, July 1989.
3. S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In *Proc. 20th Conf. on Very Large Data Bases*, pages 606–617, Santiago, Chile, 1994.
4. 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.
5. O. Diaz, N.W. Paton, and P. Gray. Rule Management in Object-Oriented Databases: A Uniform Approach. In *Proc. 17th Int. Conf. on Very Large Data Bases*, pages 317–326, Barcelona, Spain, 1991.
6. 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.
7. S. Gatziu and K.R. Dittrich. Events in an Active Object-Oriented Database System. In N.W. Paton and M.H. Williams, editors, *Rules in Database Systems.* 1st Int. Workshop on Rules in Database Systems, Springer Verlag, 1994.
8. N. H. Gehani and H. V. Jagadish. Ode as an Active Database: Constraints and Triggers. In *Proc. 17th Int. Conf. on Very Large Data Bases*, pages 327–336, 1991.
9. N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite Event Specification in Active Databases: Model and Implementation. In *Conf. on Very Large Data Bases*, pages 327–338, 1992.
10. 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.
11. 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 Conf. on Product and Process Modelling in the Building Industry*, to appear, 1995.
12. E. Simon and A. Kotz-Dittrich. Promises and Realities of Active Database Systems. To appear in Proc. Int. Conf. on Very Large Databases, 1995.

13. M. Stonebraker, a.J. Jhingran, j. Goh, and S. Potamianos. On Rules, Procedures, Caching and Views in Data Base Systems. In *Proc. of SIGMOD*, pages 281–290. ACM Press, 1990.
14. J. Widom. The Starburst Active Database Rule System. *IEEE Transactions on Knowledge and Data Engineering*, to appear.

This article was processed using the LaTeX macro package with LLNCS style