Collision of Constrained Work Spaces: A Uniform Concept for Design Interactions

Rose Sturm, Jutta A. Mülle, Peter C. Lockemann
Fakultät für Informatik, Universität Karlsruhe
D–76128 Karlsruhe, Germany, lockeman@ira.uka.de

Abstract

[1] *Since participants in a design process are almost separated in time (in addition to the usual separation in space), databases are an ideal medium to support and control their interaction. The basic premise of the paper is that each design artifact can be represented as a design object which occupies a certain domain in an n–dimensional work space which includes time in order to reflect the design stage during which certain design decisions are valid. The central hypothesis of the paper is that various kinds of interaction among designers such as detecting the effects of decisions by other designers and responding to them by, e.g., performing corrections or retracing older states in order to resume work from there, and such as detecting that own decisions may affect other designers and reacting to them by explicit notification of changes to other individuals, can all be explained by the same concept of work space collision. Constraints associated with the work spaces can be used to arbitrate between the conflicts in the colliding works spaces. The idea is applied to a variety of interaction forms to demonstrate its validity.*

1. Introduction

Design processes almost invariably engage a (small or large) number of designers. Even though the organization of a design effort is to separate it into a number of reasonably independent and often overlapping design tasks, there is nonetheless a fair degree of collaboration and, hence, interaction needed among the designers in order to guarantee that the overall result of the individual activities meets the common goal and that the individual results can be integrated into the final design product with reasonable effort.

[1]this work was supported by Deutsche Forschungsgemeinschaft, project no. Lo 296/11.

Interaction across collaborating designers is predominantly by exchanging information. If designers are separated in space interchange of information is via message transmission across communication channels. If they are separated in time – be it minutes, hours or days – databases must be interposed as the medium for the interchange of information, and interaction is initiated by changing the contents of the database and accepted by taking note of the changes.

Interaction via databases that exclusively relies on update and retrieval of information is too one–sided because it leaves the interaction entirely to the initiative of the retrieving person. Contrast this with message exchange where the person that introduces the changes takes the initiative. What is desirable, then, is to provide some mechanism that relieves a designer from the cumbersome and exhaustive inspection of the database to detect changes. Instead the database should offer mechanisms that direct the designer's attention to the more recent changes which seem of pertinence to him or her. Even further, these mechanisms should provide the designer with indications on whether these changes have an effect on his or her own current or earlier design decisions. Armed with these indications the designer should then be able to take appropriate action. Conceivable actions are immediate corrections, retracing older states in order to resume work from there, or initiating dialogues with other individuals. Alternatively, a designer who introduces changes may use these mechanisms to examine the database for effects on the work of other designers, and notify these if such effects are suspected.

The foregoing analysis already provides us with the clues to the solution: Indicate to which part of the database the attention should be directed, determine the potential for an effect on the work of other designers, if it exists determine whether there is indeed an effect in the form of some conflict and arbitrate between them, and finally permit or initiate some action.

The idea sounds fairly familiar: It bears a close kinship to the classical event–condition–action (ECA) rules known from, e.g., active database systems [33, 1]. The contribution of this paper is to adapt these rules to the area of collaborative design. It employs as its basic premise that each design artifact can be represented as a design object which occupies a certain domain in an n–dimensional work space (only part of it geometrical) including time in order to reflect the design stage during which certain design decisions are valid. The potential of conflict can then be explained in terms of a collision of work spaces. By associating constraints with the work spaces – effectively forming a constraint database – the constraints can be used to determine the actual conflicts and to arbitrate between them.

This paper is organized as follows. Section 2 introduces the main constituents of our approach while Section 3 identifies the main forms of interaction that are to be supported. Section 4 develops the main support mechanisms of work space collision and rule handling. Section 5 shows how these are combined into the core concepts of descriptively formulated queries and updates. Section 6 deals with the issue of rule execution, and applies the core concepts to the issue of backtracking of design decisions. Sections 7 places our approach in the context of other work on collaboration support, while Section 8 reports on our experiences with the approach in an application of building design, and draws some conclusions.

2. Constituents

2.1. Work spaces

Intuitively speaking it seems fairly obvious that a design artifact occupies a physical space which must be reflected in the corresponding informational design object, and that the design object has an additional temporal dimension which reflects its design stage and hence, the period during which certain design decisions were or are valid. In other words, each design object occupies a certain domain in space and time, which we refer to in the sequel as a *work space*. Our cooperation with architects on database support for the design of buildings has taught us, though, that there are considerable more dimensions to such a space – and this is probably true for other engineering activities as These dimensions are due to certain characteristics of the creative design process [19, 20, 29].

- Design processes vary in their level of detail and precision. Solutions are first of a sketchy nature, for example while the architect tries out different placements of the building. Once the solution is accepted, more detail is added in a stepwise fashion. For example, the roofing material is determined, the physical principle for the sewage is settled, e.g., a sufficient gradient for natural flow versus pumping. Given the outline of the building, the layout of the floors is added, and in doing so each floor is considered separately and perhaps on a smaller scale in order to place walls, power wiring and outlets, water pipes and outlets. The level of precision is increased by fixing, e.g., three–dimensional geometries and two–dimensional placements.

- Design processes are iterative. Decisions are often tentative or even of a hypothetical or probing character. They are tested and explored for their consequences, and are as often rescinded as they are accepted. Take a building that is to be placed in terrain with a difficult topography, where the total floor space is prescribed but there is some latitude as to the exact outline and position, and where the architect exploits this range so as to minimize the cost for the foundation, water supply and sewage. In general, the iteration crosses varying levels of detail or precision.

- Design processes are view–sensitive. Each design expert, e.g., architect, public agency, utility company, structural engineer, sanitary engineer, air condition engineer, concentrates on some particular part or aspect of the building, has his or her own particular view of the building, observes his or her own set of exterior factors, and takes decisions from his or her own viewpoint.

In our example, these characteristics are reflected by a multi–dimensional work space as shown in Figure 1 and called the A4 space [19, 20]. If a design object exists in different alternatives, or on different levels of detail or precision, it gives rise to different spaces that often overlap in time or space. If new decisions are taken and documented as such, the space is not overwritten but rather a new one is added with a new life span.

The work spaces of design objects (henceforth referred to as *object work spaces*) reflect results (or snapshots) of the design process. The process itself is under the control of the designer who must be able to move freely between any design object or arbitrary collection of objects, perhaps on different levels of abstraction, and perhaps even back in time. In order to capture the movement, each designer should be able to specify his or her current domain of interest, this again in terms of the A4 space. We refer to such a domain of interest as the (current) *user work space*.

x, y, z	intervals of numbers	geometry of bounding box of decision space
t	interval of numbers	life span (temporal validity) of decision space
scale	quotient of numbers	level of detail
resolution	value from enumeration	precision level
association	string value	building section affected by the decision
role	string value	building functionality affected by the decision
version	number	current alternative if several considered in parallel
user	string	decision maker
composition	list of identifiers	list of parts
meta	string	additional descriptive information

Figure 1. A4 work space for the design of utility buildings

2.2. Events

If we now attach work spaces to rules, a first indication of conflict and one that allows to leave the initiative with the designer initiating a change is to place an event in a spatial dimension over and above the usual temporal dimension. In our example, events must be associated with A4 spaces, and are associated with a subdomain delimited by exact values for the spatial coordinates x, y, z and the time coordinate t, but are open on the remaining dimensions.

2.3. Constraints

Design processes can be interpreted as alternating between narrowing and widening of a decision space, with an overall tendency towards narrowing until no decisions have been left open. Hence, design objects become more concrete as they evolve over time. In building design, only few factors exist at the beginning that constrain the decision space, such as topography, intended utilization, municipal water lines and sewers. Subsequent design decisions add further constraints. These may take the form of exact values as for the roofing material or the overall metrics for the building, but more often they involve a reduction of ranges of possible values such as the positions of outlets. Certain constraints exist over the entire design cycle, however, such as zoning regulations, building standards, laws and regulations on fire protection, environmental protection, use of hazardous materials, working conditions, parking spaces. Likewise to be observed are physical laws, such as air circulation, minimal gradients of plumbing for sewage, weights and needed support, as well as the multitude of standards on materials, prefabricated components, wiring fuses, to name a few.

Consequently, a design space is circumscribed by a number of constraints – external factors and earlier design decisions [19, 20, 29], and a constraint is valid in one or more work spaces. A conflict arises in fact if a constraint becomes violated in a work space due to a design decision taken by the proprietor of the space or by a collaborating designer.

The notion of work space can also serve as vehicle to group constraints. For example, different experts with their different objectives and views will in all likelihood contribute their own specific and often predetermined set of constraints. We may collect such a set into an A4 space (*constraint space*), where the constraints are assembled into the composition dimension while most other dimensions remain open.

3. Interaction

3.1. Retrospective interaction

Retrospective interaction refers to interaction that is entirely due to the initiative of a designer who did not originate certain changes but may be forced to react to them. Retrospective interaction must, therefore, rely on the inspection of the design database. As mentioned in the introduction, this is a cumbersome affair as long as one must exhaustively search the database for recorded changes. It becomes much more efficient if one could perform a selective search. The traditional means for such a search is *querying* the database in order to circumscribe a certain sphere of interest where an interaction could have a potential effect.

As noted in Section 2.1, such a sphere of interest can be captured by the concept of user work space. In turn, the design objects that may potentially have an effect are the ones whose object work spaces somehow overlap the user work space or, as we say, *collide* with the user work space.

3.2. Prospective interaction

By contrast, prospective interaction refers to interaction that relies on the initiative of the originator of a change. It is his or her responsibility to draw the attention of other designers to the change. If the affected designers are unknown a kind of broadcast mechanism must be pursued in place of a more direct message exchange. As mentioned in the introduction, the traditional means in databases is for the originator to raise an event. In turn, designers who expect to be affected by the changes should *subscribe* to the corresponding events.

In our approach, subscription is handled by a combination of two facilities. The first is to identify the types of events that one may potentially have to respond to. The second is to select among the raised events only those that fall within the current sphere of interest, i.e., the current user work space. By assigning events a spatial and temporal dimension (Section 2.2), events are easily associated with work spaces. The association can be viewed as a special case of work space collision where the affected work spaces are those that overlap the event space (usually a point space). Within each work space one may then identify the design objects that merit close inspection because their work spaces collide with the event space.

3.3. Reaction

Even after those object work spaces have been identified which merit closer attention, the detection of the changes of importance may still require meticulous comparisons. The designer would gain help if the database system pointed out to him or her where changes occurred that affect his or her own decisions. A standard mechanism for this purpose is checking the constraints in his or her user work space – his or her own decisions – for violations due to the decisions of other designers.

Constraint checking must take into account, though, that due to the tentative character of design decisions the designer will tolerate at times a certain degree of contradictions among decisions or between decisions and externally imposed constraints such as regulations or availability of needed resources. Consequently, the designer must be able to exercise control over the tolerance level.

Once a violation has been detected the designer must take some action. Actions may range from entering into a dialogue with the originator of the change to an independently performed repair. One particularly interesting repair action is the restoration of an earlier state of work spaces by rescinding certain decisions (selective backtracking). Taking back decisions may again have to be confined to a specific work space. For example, consider a floor with its placement of walls, power wiring and outlets, water pipes and outlets. Conflicts between wiring and pipes may force the architect to rescind some of his or her earlier decisions. Hence, a selective backtracking action can again be expressed in terms of work space collision.

Backtracking may in turn cause new interactions because the changed work spaces may affect the work spaces of other designers who may have to employ querying or constraint checking.

4. Interaction mechanisms

4.1. Collisions

Work space collision occurs if the values or intervals overlap in at least one of the dimensions. Hence, what constitutes a collision depends on the definition of overlap for the various dimensions. The definition clearly is different for continuous and discrete dimensions, and one must distinguish between ordered and unordered values. Take ordered dimensions where one may define some sort of interval arithmetic. Its operators depend on the value type. For example, overlap of intervals is handled differently depending on whether one deals with real numbers, integers, or time based on some calendar system. For unordered values there is no notion of interval so that overlap is simply defined as the coincidence of values.

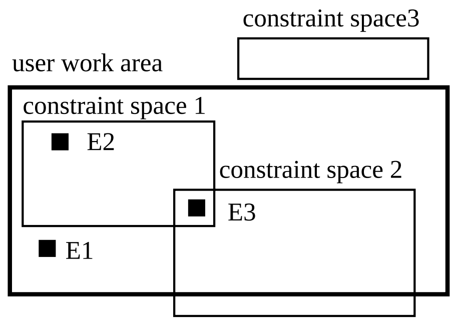

Figure 2. Collision of events with work spaces

The collision concept will be illustrated in the remaining sections. For a precise treatment see [30]. We use simplified two–dimensional spatial arrangements for work spaces so that a collision can be depicted as a simple spatial overlap. As a first example, though, consider the collision of event spaces with work spaces.

We also show that the definition of what constitutes a collision depends on the particular circumstances and concomitant reaction.

In Figure 2 an event (event_type, position) has a point space in the user work space. Spaces 1, 2 and 3 are spaces associated with three different constraints, and E1, E2 and E3 are events arising in the shown work area. space3 lies entirely outside the work area and will never be affected by events within the user work area. E1 does not collide with any of the other two constraint spaces and, hence, will not cause any constraint check. E2 collides with space1 and will cause further examination of the corresponding constraint (provided its underlying event type agrees with the event type of E2). Likewise, the rules of both space1 and space2 are candidates for a check on the occurrence of E3. That space2 does not completely fall within the work area is of no significance here. Now suppose that in our case study spaces 1 and 2 represent two zones in a building together with the layout of their pipes for the water supply, with the overlap symbolizing a vertical shaft for the water main. Suppose further that the designer redesigns the shaft by, say, enlarging the diameter of the water main. Then there is at least a possibility that piping in the two zones must be revised.

Let event E1 signify that the designer moves an electricity outlet somewhere outside the two zones. If it was outside before as well, then clearly there is no effect on the two zones. If it was inside, though, it is no longer obvious that piping in the two zones remains unaffected. Hence, another interpretation of events seems better suited, namely one which associates with an event the entire work space, i.e. treats the event as a tuple (event_type, work_space). In such a case event E1 indiscriminately affects spaces 1 and 2. We notice again that we must provide even within the same application more than one way on how to define and deal with collisions.

4.2. Rules

Rules codify the subscription mechanism for prospective interactions: They must identify the event types to which a designer wishes to pay attention to, the work space within which an event of the given type indicates a potential effect, a constraint that would have to be checked to determine whether such an effect does indeed exist (thus in essence mediating between the work spaces and, hence, designers involved), and specify an action to be taken (which may range from notifying the user of the result of the constraint check to invoking some automatic reaction). Figure 3 shows the general format, basically an event–condition–action (ECA–) rule which has been extended by a spatial clause.

in	work space
on	set of event types
if	constraint
do	set of actions

Figure 3. Rule format.

Rules can easily be extended to deal with retrospective interaction as well: Associate with queries a special event type, and raise for a given query an event (event_type, user_work_space).

Since each constraint is associated with a constraint space within which it is valid, and such a space may cover a number of object work spaces, rules are also the means to describe constraints. Essentially the declaration of constraints and subscriptions are merged into a single concept.

5. Queries and updates

5.1. Queries

Queries are the basic mechanism for retrospective interaction. As noted before, in our environment queries are characterized by specifying a search work space (query space), e.g., a user work space, and are resolved by determining the object work spaces with which the query space overlaps. This approach corresponds to a descriptive formulation of set–oriented retrieval well–known from database systems [9]. The view of query processing as the detection of a collision of spaces also bears a close relationship to the various data structures and algorithms that support multi–attribute queries such as grid files or multi–dimensional trees [28]. Particularly illustrative are structures that directly support spatial queries in connection with geographic information systems [22]. The similarity also suggests that the access path techniques developed for multi–attribute queries could easily be adapted to query processing that reflects the collision concept.

Figure 4 gives a simple visualization. In it the query space collides with three object work spaces. Several kinds of query processing are conceivable. For example, all colliding objects may be recovered, or only those that entirely lie within the query space, or all objects that are covered by the query space to more than 50% (however this is computed).

Indeed there is a more general aspect to our approach: Anything that can be represented as a work space can be the subject of retrieval by a query, with a specific choice of the type of collision processing. This introduces an element of orthogonality into query processing. For example, since a query is itself determined by a space a query could be used to recover a set of pertinent queries. Queries could be used to examine constraint spaces. Constraints could be checked on queries, or be attached to queries.

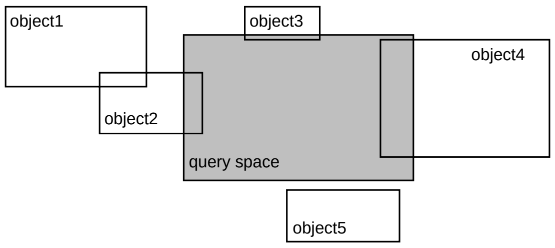

Figure 4. Visualization of query processing

5.2. Updates

The ultimate source of all interaction are updates to the design database. Since a designer operates within a work space, his or her updates are confined to his or her current work space. Updates may not just set values on one or more dimensions but may combine work spaces, as when the designer wishes to attach a predefined constraint space to his or her current work space, effectively merging the two. The current work space may overlap with the current or potential work spaces of other designers and thus defines the potential for interaction.

In set–oriented descriptive languages updates resemble queries and indeed have in common with them an identification of objects, in this case those that are to undergo changes. Consequently, in our environment updates designate a query space which in turn determines by collision the objects to be changed. For example, to delete object work spaces from the database, one provides a query space that contains them. To modify for selected work spaces the values along certain dimensions one follows the same selection procedure. Insertion of a new space, though, falls outside the scope of the collision mechanism.

In the case of prospective interaction, one would also have to raise corresponding events. One could also take a more precautionary action by using the same query to identify work spaces of other designers with a potential for conflict due to constraint violations.

Due to the temporal dimension of work spaces, major design decisions result in all likelihood in a new decision space for the affected objects. Deleted objects must be preserved if there is a chance that their space may become the target of later backtracking in history. Modifications and insertions may later have to be rescinded, hence the old work space must be retained and a new one constructed. In our case study, a corresponding versioning mechanism sets for an old space the upper boundary of the t–interval to the time of change, and for a new space the lower boundary to that time value. Figure 5 illustrates in successive steps of insertion, deletion and modification the situations with constraint spaces (note that for the sake of illustration the space has been split into geometric area and time interval).

6. Selected issues

6.1. Rule execution options

As noted in section 4.2, subscriptions and constraints are treated uniformly and are maintained in the form of ECA rules tied to some validity space. Rules, or triggers, are known from active database systems as a means for constraint checking [5, 6, 7, 33]. Consequently, one would expect to carry over the execution semantics and facilities from these systems.

However, there are some differences. To the best of our knowledge the use of constraint checking by rules as a means for controlled interaction is new (it was first discussed by the authors in [29]). It is reflected in the rules by including work spaces as a separate clause rather than as part of the constraint clause. This extension even provides some technical benefit: Searching the rules affected by a given event is more limited and, hence, much faster. This particular aspect has also been a motive in the association of rules with contexts in SAMOS and Sentinel. SAMOS takes contexts into account by supporting event parameters that restrict complex events to relevant categories such as a specific transaction or a particular user [10]. Sentinel provides rule evaluation contexts which differ in the set of relevant events contributing to complex events [8].

Traditionally, constraint checks and, hence, rules may be activated or dectivated [1]. Since we include the dimension of time in our notion of work space, activation and deactivation is simply achieved by defining a new space (with the time dimension changed) with each such constraint because collision takes place only for those spaces whose time dimension collides with the time of the event.

By manipulating the time of an event one can achieve interesting effects. For example, since according to section 5 we maintain the history of spaces, one could explore which effects a given event might have

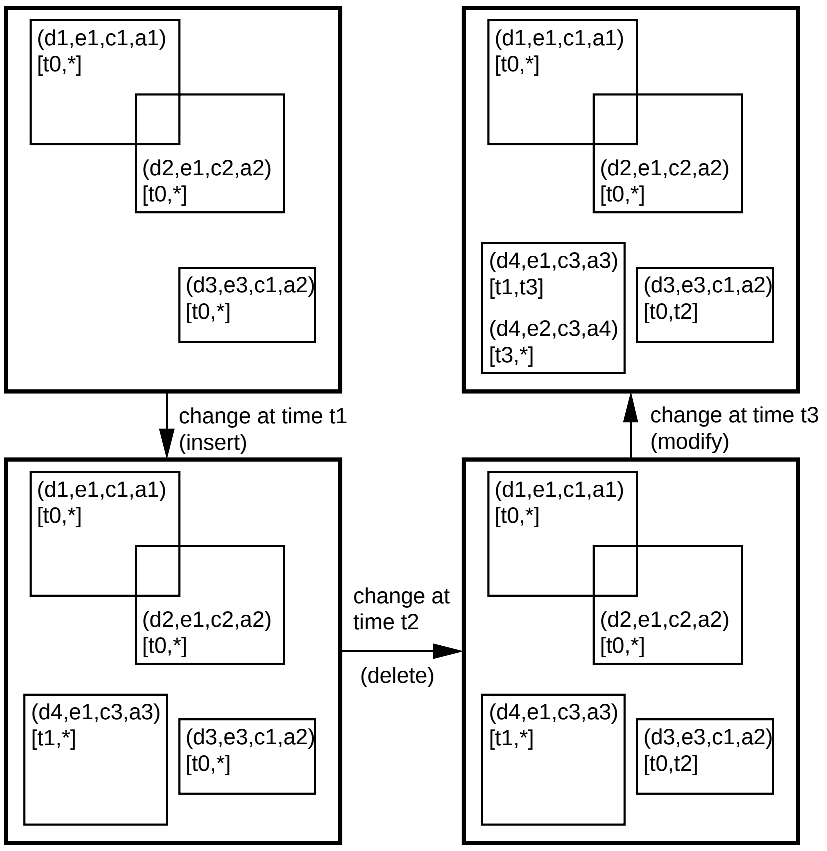

Figure 5. Successive states of a constraint space after insertion, deletion and modification (in that order). Letter codes are d for area, e for event, c for constraint and a for action; * is open boundary. The time intervals are shown separately.

had in the past in order to find out how many inconsistencies were tolerated at some earlier time. Or one could hold an event for some time in order to execute it once certain constraints have been reactivated, although to carry events over into the future requires some persistence mechanism for events.

After an event was recognized to apply to an ECA rule, the time at which its condition and action parts are executed can be controlled by coupling modes [1]. This has been uncritical so far because the the only type of action we allow is notification of the user. We permit a variety of responses when a check of the condition fails: No notification (ignoring the check), notification only if explicitly desired, immediate notification and continuation of regular processing, immediate notification with interruption, refusal of the user activity that gave rise to the event. Clearly then, the tolerance level for inconsistencies can be adjusted by updates to the rule via either or both the time dimension of its work space and the action.

6.2. Selective backtracking

Backtracking has a long tradition in databases. Its main use is to maintain the atomicity of transactions [3]. A single instance of the past, the before–image, is kept until the transaction has successfully completed, then it is discarded. If one employed a versioning mechanism, more of the history remains available. Therefore, Lausen [18] extended versioning to databases. By far the closest to our intentions as far as the temporal dimension is concerned are temporal databases [31, 23, 27, 32], since one may view backtracking as querying the past and, as shown in section 5.1, collision is a form of querying.

Since our notion of space is multi–dimensional, we can use more than the time dimension for backtracking and thus be both, more general and more selective than traditional backtracking. In principle, the selection can again be done by collision: Define a backtracking space, and then select all spaces which collide with this space.

Figure 6 illustrates the concept for a t4, $t0 < t4 < t1$.

In the figure all four constraint spaces collide with the backtracking space. Three of them collide on time and geometry. Spaces d1 and d2 remained the same over the entire time range and thus are not changed by backtracking so that even the fact that the backtracking space splits d1 has no effect. In space d3 the upper boundary has been reset to the open boundary "*" because at time t1 the update time t2 must still have been unknown. Space d4 collides on geometry alone so that it merits closer inspection.

Clearly, the area within the backtracking space is no longer valid. But what about the area outside the backtracking space? The specific actions to be taken are highly application–dependent. For example, in our case study we distinguish between closed constraints which if valid in a space are valid in each subspace, and open constraints if otherwise. Physical laws are closed constraints, whereas a design decision such as that the room size must not fall below a certain value is not. For open constraints the only meaningful solution is to roll back the entire space, even the portion outside the backtracking space. In such a case d4 would have to vanish in its entirety. For closed constraints, however, it may make sense to leave the area outside the backtracking space intact and to roll back only the portion inside. Figure 6 illustrates this case (the outside portion has been renamed d5). One somewhat confusing result of selective backtracking is that the overall design space is composed of spaces with different validity.

7. Related approaches

To summarize our approach, interaction is determined dynamically by employing work space collision and constraint checking to establish the potential and the actual interaction, respectively. We leave a high degree of freedom to the strategy for the desired collaboration. One may choose arbitrary dimensions for the work spaces, associate different semantics with what constitutes a collision, allow different interpretations for a conflict, decide when to activate and deactivate rules and choose among different coupling modes and different options even for standardized corrective actions. By deciding on the order and concurrency of event processing an event handler can impose its own strategy on the collaboration. Likewise, the application of events into the past or the future and selective backtracking of constraints would have to follow a strategy accepted and understood by all participants.

One may view our approach as a generalization and amalgamation of various other approaches Two were already mentioned above, active databases and versioned or temporal databases. Shared work spaces are also a means for computer–supported interaction in CSCW where each participant in collective work may directly observe the results of others [12, 14, 16, 4]. Limitations are the rather small set of data, that can directly be observed, and the fairly rigid discipline on how to resolve conflicts and how to pass control among the collaborating partners, usually with a low degree of concurrency.

Interaction by detection of constraint violations could be viewed as some sort of management–by–exeption. In it, interaction is mainly corrective in that conflicts must be recognized as soon as they occur, and some automatic action triggered or the situation communicated to the participants. It is a typical solution to the management of long–living transactions in shared design databases [13] but also plays a role in CSCW [17]. Again, these solutions impose a fairly rigid discipline on the collaboration in order to meet the overriding goal of consistency, and if they consider history this rarely goes beyond the needs of transaction recovery or compensation.

Shared work spaces, though limited to geometry, also seem a natural approach to simultaneous engineering. And indeed, this is an approach taken by [11] where cooperation semantics may be formulated with the aid of interfaces to the spaces. Such semantics may also be based on common product models for the spaces [15, 24]. Other approaches to structuring work work spaces are found in [26] with multiple views on objects: the shared part of visualization data, the underlying representation, and organizational data about, e.g., versioning, access rights; in [2] with relationships between different application protocols; and in [25] and [21] with communication relationships between distinct databases and views.

8. Implementation and Experiences

Our approach was developed within a joint project with architects on database support for concurrent design of utility buildings that conform to a fairly rigid standard in terms of rastered layout and building components. Architects, utility companies, structural engineers, sanitary engineers, air condition engineers, among others, must collaborate under conditions of short development cycles where much of the experts' work must go on concurrently. Design conflicts must either be avoided or be brought out in the open immediately so that they can be cleared up. Just consider how wiring and plumbing affects the placement of interior walls and vice versa.

For the CAAD environment we use a three–layered

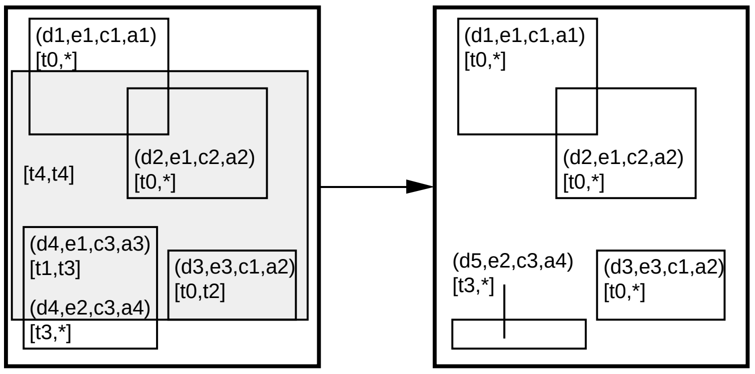

Figure 6. Selective backtracking to time t4, $t0 < t4 < t1$. Notation is as in Figure 5.

implementation architecture. The topmost layer – the representation layer – reflects the user's perceptions of the design process, design objects and design spaces. An elaborate user interface developed by the architects allows users to deal with all dimensions of the A4 space, not just with the geometrical confines. In particular, events, constraints, actions may be visualized. In case of collaboration the overlap of spaces may be visualized by popping in on the screen the foreign constraints that may be affected by one's own actions. The next lower level implements the execution engine for event detection, collision handling, constraint checking and all the subsequent actions. The lowest level provides the implementation platform. It includes an object–oriented database management system, and performance enhancing data structures such as multi–dimensional indexes for identifying the colliding spaces.

Considerable work especially for plumbing and wiring was delegated to expert system programs. This was a major motivating force in the direction of formalizing the design process and coming up with a sufficiently uniform model. One of them, the MIDI expert tool [34], covers all "room-building" components, e.g., structure, facade, interior walls, and wiring. Design constraints were jointly developed with the architects and stored in the database. A user interface for interacting with the constraints was added to MIDI. The system was then applied to design tasks for multi–storied office buildings. Response times in these examples were in the range of less than a minute for the constraint checks. This is partly due to the spatial locality of the constraints that results from the modelling of constraints in work spaces.

The examples so far have not been extensive enough to render final judgment on how well our concept would scale up. Nonetheless, we believe that a major attractiveness of the collision concept lies in its orthogonality to a variety of situations and the strict separation of a uniform basic collision recognition mechanism from a collision processing facility which may be subjected to a strategy tailored to the demands of a particular situation.

References

[1] The ACT–NET Consortium. *The Active Database Management System Manifesto: A Rulebase of ADBMS Features.* To appear in ACM SIGMOD Record 25 (1996), No.3

[2] R. Anderl, A. Wasmer. *Methoden zur Modellintegration im Produktentwicklungsprozeß.* it+ti, 37(5), 1995, 18-23

[3] P. Bernstein, V. Hadzilacos,and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison–Wesley 1987

[4] M.P. Case, S.C.-Y. Lu. *Discourse Model for collaborative design.* Computer-Aided Design, 28 (5), 1996, 333-345

[5] S. Ceri, J. Widom. *Deriving production rules for constraint maintenance.* Proc. 16th Int. Conf. on Very Large Data Bases 1990, 566-577

[6] S. Ceri, P. Fraternali, S. Paraboschi, L. Tanca. *Constraint enforcement through production rules: Putting active databases to work.* IEEE Data Engineering 15 (1992), 1-4, 10-14

[7] S. Ceri, P. Fraternali, S. Paraboschi, L. Tanca. *Automatic generation of production rules for integrity maintenance.* ACM Trans. Database Syst. 19 (1994), 367-422

[8] S. Chakravarthy, V. Krishnaprasad, E. Anwar, S.-K. Kim. *Composite Events for Active Databases: Semantics, Contexts and Detection.* Proc. 20th Int. Conf. on Very Large Data Bases 1994, 606-617

[9] C. J. Date, H. Darwen. *A Guide to the SQL Standard.* 3rd ed. Addison–Wesley 1993

[10] S. Gatziu, K. R. Dittrich. *Events in an Active Object–Oriented Database System.* In N.W.Paton, M.H.Williams (eds.): Rules in Database Systems. Workshops in Computing Series, Springer 1994, 23-39

[11] H. Grabowski, M. Schmidt. *Distributed Design – Working in Design Spaces (in German).* Prod. CAD'92 GI Workshop, Berlin, Springer 1992, 219-232

[12] I. Greif (ed.). *Computer–Supported Cooperative Work: A Book of Readings. Part II: New Technologies for CSCW.* Morgan Kaufmann 1988

[13] S. Heiler, S. Haradhava, S. Zdonik, B. Blaustein, A. Rosenthal. *A Flexible Framework for Transaction Management in Engineering Environments.* Ch. 4 in A.K.Elmagarmid (ed.): Database Transaction Models for Advanced Applications. Morgan Kaufmann 1991, 87-121

[14] H. Ishii, N. Miyake. *Toward an Open Shared Workspace: Computer and Video Fusion Approach of Teamworkstation.* Comm. ACM 34:12 (1991), 37-50

[15] P. Katranuschkov. *COMBI: Integrated Product Model.* In R.J. Scherer (ed.), Proc. 1st European Conf. on Product and Process Modelling in the Building Industry, October 1994, Dresden, A.A. Balkema 1995.

[16] S. Khoshafian, M. Buckiewicz. *Introduction to Groupware, Workflow, and Workgroup Computing.* John Wiley & Sons 1995

[17] M. Kyng. *Designing for Cooperation: Cooperating in Design.* Comm. ACM 34:12 (1991), 65-73

[18] G. Lausen. *Formal Aspects of Optimistic Concurrency Control in a Multiple Version Database System.* Information Systems 8 (1983), 291-301

[19] P.C. Lockemann, J.A. Mülle, R. Sturm, V. Hovestadt. M*odeling and integrating design data from experts in a CAAD environment.* In R.J.Scherer (ed.): Product and Process Modelling in the Building Industry. A.A.Balkema 1995, 29-34

[20] P.C. Lockemann, R. Sturm, J.A.Mülle, V. Hovestadt. A*rea–dependent constraints for design control in a CAAD environment.* In P.–J. Pahl, H. Werner (eds.): Computing in Civil and Building Engineering. Vol.1. A.A.Balkema 1995, 755-762

[21] B.K. MacKellar, J. Peckham, M. Doherty. *A data model for the extensible support of explicit relationships in design databases.* The Very Large Database Journal, 1995

[22] B. C. Ooi. *Efficient Query Processing in Geographic Information Systems.* Lect. Notes in Computer Science 471. Springer 1987

[23] G. Ozsoyoglu, R. T. Snodgrass. *Temporal and Real–Time Databases: A Survey.* IEEE Trans. on Knowlege and Data Engng. 7 (1995), 513-532

[24] M.A. Rosenman, J.S. Gero. *Modelling multiple views of design objects in a collaborative CAD environment.* Computer-Aided Design, 28 (3), 1996, 193-205

[25] M. Rusinkiewicz, A. Sheth, G. Karabatis. *Specifying Interdatabase Dependencies in a Multidatabase Environment.* Computer, December 1991, 46-53

[26] M. Saad, M.L. Maher. *Shared understanding in computer-supported collaborative design.* Computer-Aided Design, 28 (3), 1996, 183-192

[27] R. T. Snodgrass (ed.). *The TSQL2 Temporal Query Language.* Kluwer 1995

[28] M. Stonebraker (ed.). *Readings in Database Systems.* 2nd ed. Chapter 2: Relational Implementation Techniques, Morgan Kaufmann 1994

[29] R. Sturm, J. A. Mülle, P. C. Lockemann. *Temporized and localized rule sets.* In T. Sellis (ed.): Rules in Database Systems. Lect. Notes in Computer Science 985. Springer 1995. 131-146

[30] R. Sturm. *Dynamic Rule Sets Describing Design Lattitudes (in German).* To appear: VDI Verlag, 1997

[31] A. U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, R. Snodgrass. *Temporal Databases: Theory, Design and Implementation.* The Benjamin/Cummings Publ. Co. 1993

[32] V. J. Tsotras, A. Kumar. *Temporal Database Bibliography Update.* ACM SIGMOD Record 25 (1996), No.1, 41-51

[33] J. Widom, S. Ceri. *Active Database Systems.* Morgan–Kaufmann 1995

[34] H. Wichmann (ed.). *System-Design: Fritz Haller Bauten - Möbel - Forschung.* Birkhäuser Berlin, 1989 (in German)