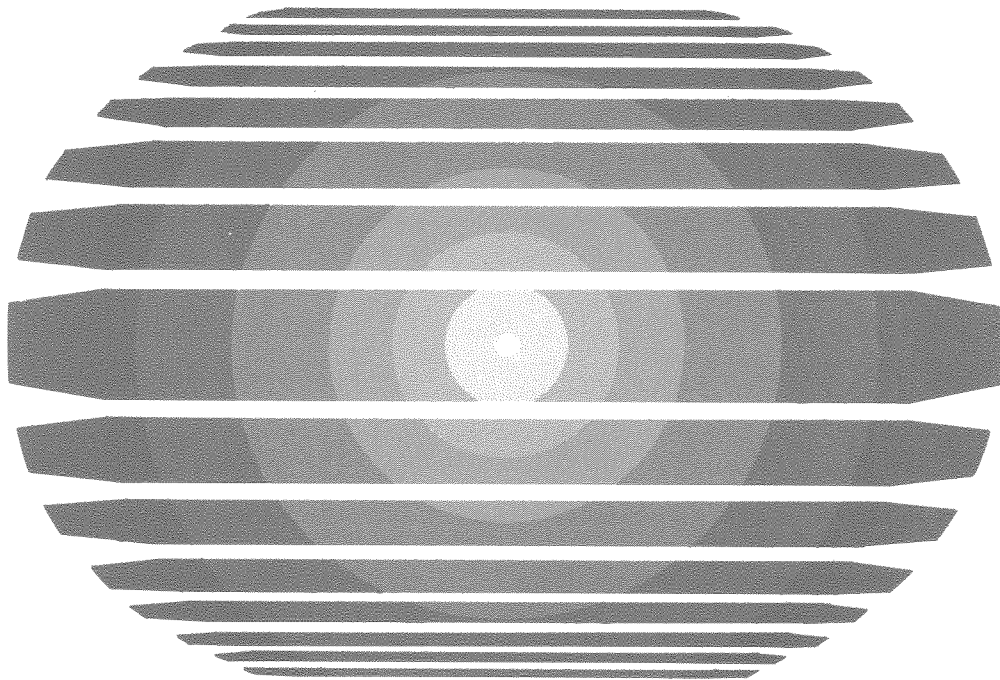
Rio Scientific Center

Technical Report CCR-118
December 1990

An information system environment based on plan generation

Antonio L. Furtado
Marco A. Casanova



73010

Rio Scientific Center

Technical Report CCR118

An information system environment based on plan generation

by

Antonio L. Furtado [1,2]

Marco A. Casanova [1]

[1] Centro Científico Rio
P.O. Box 4624
20.001 - Rio de Janeiro - RJ
Brazil

[2] Departamento de Informática
Pontifícia Universidade Católica do R.J.
Rua Marquês de S. Vicente, 225
22.453 - Rio de Janeiro - RJ
Brazil

ABSTRACT

The purpose of the research project outlined in this paper is to investigate knowledge based methods and to develop software prototypes that provide a *cooperative environment* for the use of information systems. The paper concentrates on three distinguishing characteristics of the project: the use of *rules* and *clauses* to specify all aspects of the environment, including operations; the use of a *temporal database* recording past, present and future states; and the use of a *plan and schedule generation algorithm* handling conjunctive goals and producing plans consisting of sequences of actions.

1. INTRODUCTION: TOWARDS COOPERATIVE ENVIRONMENTS

The purpose of project **NICE**, to be outlined in this paper, is to investigate knowledge based methods and to develop software prototypes that provide a *cooperative environment* for the use of information systems. An environment is cooperative [BJ,CC,We] to the extent that it helps the user to interact with an information system in ways that:

1) contribute to the achievement of the user's goals;
2) keep the user's understanding of the system in harmony with the definition and contents of the system, avoiding misconceptions and contributing to a fuller and at the same time more efficient usage;
3) conform to the established integrity and authorization constraints.

Such an environment can also be interpreted as a cooperating system, in the sense of [De], by regarding the information system and each user as the distinct agents of the cooperating system.

We will concentrate on three distinguishing characteristics of the project: the use of *rules* and *clauses* to specify all aspects of the environment, including operations; the use of a *temporal database* recording past, present and future states; and the use of a *plan and schedule generation algorithm* handling conjunctive goals and producing plans consisting of sequences of actions.

The paper is organized as follows. Section 2 shows how different classes of rules are applied to modify a request. Section 3 introduces the syntax of the clauses used to specify the time-dependent facts and operations. The structures we use to implement temporal databases are mentioned in section 4. The central component of the present prototype - the plan and schedule generation algorithm - is the object of section 5. Some examples of cooperative behaviour are given informally in section 6. Section 7 contains the conclusion. The appendix contains further examples of the concepts introduced throughout the paper. Finally, references to and comparisons with the relevant literature will appear throughout the sections.

2. THE USE OF RULES TO MODIFY A REQUEST

In what follows, we refer to a query or update command as a user *request*. To achieve cooperative behaviour, the system cannot execute a request literally, but rather it has to modify the request appropriately. Although request modification features are not new in the database area [St], they have been used for the specific purpose of enforcing integrity and authorization rules, being applied only before request execution.

The instruments we use for this purpose are rules, classified according to their generality as *domain-independent* rules, that originate from the conceptual model adopted to describe databases, and *domain-dependent* rules, that are specific to the application domain in question. Rules are also classified according to the phase at which they are applicable: *"pre"* rules are applied before the execution of a command to *correct* or *complement* a request; *"s_post"* rules are applied after successful execution to *complement* a request or to *articulate* an answer; *"f_post"* rules are in turn applied after execution in case of failure.

We now elaborate further on the nature of the rules. Complementing a request means doing more than has been asked, that is, supplying additional information (for queries or updates), or executing additional changes (in case of updates). "Pre" (resp. "s-post") rules to complement a request arise when it is possible to detect such needs before (resp. after) executing a request. It is also the job of "s_post" rules to *articulate* the information to be communicated to the user in an understandable format, perhaps omitting uninteresting or non-authorized items.

Failed executions require a more careful analysis. First observe that a "yes/no" query fails if it has a negative answer, and that a "which" query fails if there are no values satisfying the qualification. An update request in turn fails if it cannot be executed for one or more of the following reasons: its effects already hold, total or partially; or some pre-condition to its execution does not hold; or its execution would violate some constraint. When a request fails, the failure may be due to a *removable* obstacle, consisting of pre-conditions that do not hold when the request is posed but that will perhaps hold in the future. If so, "f_post" rules may propose that the system's monitor will *alert* the user when the obstacle ceases to exist. They may instead (or if the obstacles are known to be of a persistent nature) generate and execute an *alternative* request that accomplishes, as much as possible, the same purpose. This may take different forms that we shall not enumerate here; typically, either different values or different facts or operations may be involved. If no alternative exists or if all alternatives fail, "f_post" rules can at least proceed to *explain* the reason of the failure.

Rules are typically of the form:

```
for request R, if condition C holds then perform action A
```

The most common action in `A` is to transform `R` into a modified request `R'`. Condition `C` refers, of course, to the parameters of `R`, but it may also be based upon a context including the following components (see Figure 1):

- the database conceptual and external schemas
- the factual database

- a model of the application domain
- the users' profiles, including, for each user, his often faulty understanding of his external schema
- a repertoire of typical plans, either pre-established by the system designers or recorded from past interactions with users
- a record of the ongoing session, registering requests from and responses to the current user

Note that this environment is far richer than that traditionally contemplated. Thus, for example, one may design a rule that complements a request based on a match between the current sequence of actions performed by the user with a typical plan stored in the system.

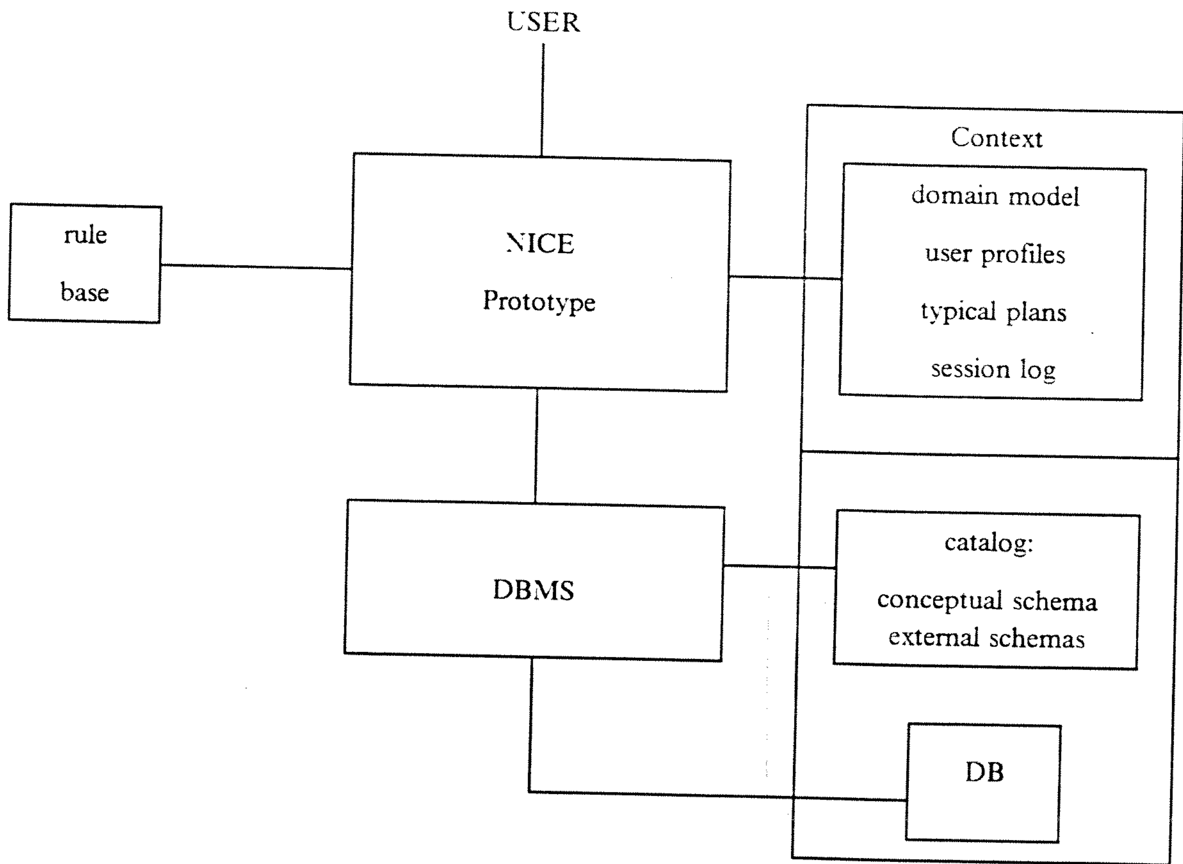


Figure 1: Architecture of the Prototype.

3. CLAUSAL SPECIFICATIONS

We adopt a uniform way to specify the conceptual schema, the operations, the integrity constraints and each database state through ground unit clauses. The appendix contains examples of the concepts introduced in this section.

Briefly, the conceptual model follows the basic entity-relationship model, extended with is-a hierarchies. The clauses that define the conceptual schema of a database then indicate which entities exist and how they are interrelated. Their syntax is the following:

```
entity(<entity class>,<key>)
is_a(<entity class>,<entity class>)
relationship(<relationship class>,[<participant classes>])
attribute(<entity class>,<attribute>)
attribute(<relationship class>,<attribute>)
domain(<attribute>,<description of possible values>)
```

A database state for a conceptual schema is a set of facts, where a fact indicates the existence of either an entity or a relationship instance, or captures that one such instance has a certain value for an indicated attribute. The syntax for facts is:

```
<entity class>¢<key>
<entity class>¢<key>\<attribute>(<value>)
<relationship class>#[<participants list>]
<relationship class>#[<participants list>]\<attribute>(<value>)
```

where <participants list> is a list of pairs of the form <entity class>¢<key>. Furthermore, the notation

```
<fact>@<time instant>
```

will be used to indicate that a fact (of one of the kinds above) holds at a specified time instant. The *database state* at time t is then the set of facts associated with t.

In the spirit of abstract data types, the operations allowed on a database are predefined. Each operation O is specified again by a set of clauses, which indicate: the facts that are added and deleted by O (i.e., the effects of O); the pre-conditions for the execution of O, in terms of facts that should or should not hold; and, optionally, lower and upper time bounds for the execution of O. Pre-conditions can be seen as a dynamic way to enforce integrity constraints, in the sense that they restrict the application of operations. The syntax of the clauses to specify operations is:

```
added(<fact>,<operation>)
deleted(<fact>,<operation>)
```

```
precond(<operation>,<expression involving facts>)
time_bounds(<operation>,<earliest time>:<latest time>)
```

Finally, integrity constraints may also be declared, without referring to specific operations, through clauses of the form:

```
imposs(<expression involving facts>)
```

Such clauses indicate conjunctions of facts that characterize states that are not valid and therefore should not be reached by any operation or sequence of operations.

The clausal specification of operations induces a network structure relating facts and operations by means of effects and pre-conditions, illustrated by the following diagram:

```
        pre-condition              effects
fact1 --------------> operation -------------> fact2
```

One may enter the network at the point corresponding to the request and then traverse it, thus moving the original focus to additional pertinent information. Recall that similar "conceptual navigation" abilities are also provided by the connections along the entity-relationship diagram, is-a hierarchies and along the time axis. In addition, the relevance of the part_of and the classification hierarchies has been advocated in [CC]. Some criteria are needed, of course, to limit traversals from the initial focus in each case. Browsing through facts and operations may show, among other things:

- if fact1 is the object of a query, this may indicate that fact1 is (part of) the user goal, but it may happen instead that it is regarded only as a possible obstacle to execute the operation in order to achieve fact2; indeed, it is often necessary to distinguish "fundamental" from merely secondary goals;
- again if fact1 is the entry point, asking only about it may mean that the user ignores that other facts are also pre-conditions to execute the operation (one could traverse other pre-condition edges from the operation back to other facts);
- if fact2 is the object of the query, it may be that the user wrongly believes that fact2 is the only effect of the operation;
- if the request is to execute the operation, there may be other operations that achieve fact2, and have perhaps different pre-conditions and other additional effects.

Two extensions of the notation introduced so far deserve mention. Facts referring to a single attribute of an entity or relationship instance are generalized to *frames*, consisting of lists of <attribute>:<value> pairs. The syntax is:

```
<entity class>¢<key> has <frame>
<relationship class>#[<participants list>] has <frame>
```

We regard frames not as a data structure physically stored somewhere, but as a language construct to collect dynamically the information available about an entity or relationship instance. Several operations on frames are provided, including frame unification and frame generalization. Frame operations permit comparisons involving the entire descriptions of entities or relationships.

The second extension allows to handle *time intervals*, which are defined in terms of the delimiting time instants. The notation is

```
<fact> @ <begin interval>:<end interval>
```

The two extensions can be combined: individual facts as well as frames can be associated with both time instants and with time intervals.

4. TEMPORAL UPDATE-ORIENTED DATABASE

In our approach, a database is not directly stored as a set of facts, but rather it is represented by means of *update-oriented structures* [TF]. In other words, in update-oriented organizations, facts are not explicitly kept but what is recorded is the execution of each update operation, together with its *time stamp*. What facts hold at a given instant of time is inferred from these records. (Event-oriented temporal databases [KS] follow a similar strategy).

Temporal databases in general allow queries not only over the current state, but also over past states. In addition, we allow recording, with time stamp T, operations that one knows will be executed at a given future instant T. As a consequence, query and update requests can also refer to future states, which shows the power of this organization even if only direct, unmodified requests are considered. For example, queries of the types "when" ("at what time?") and "how" ("by means of what operation?") can be posed.

In a cooperative environment, one can check whether what the user wants, but is not yet true, will hold within a time margin that suits his needs, if only the future pre-recorded operations are executed. One may also identify obstacles related to time, expressed in pre-condition clauses, time bound clauses, etc., and check their status against the time-stamped information.

5. PLAN AND SCHEDULE GENERATION

To more completely exploit the clausal specification of facts and operations and the temporal database organization, a *plan and schedule generator* [FC] can be used. We define a *plan* as a sequence of operation instances and a *schedule* as a sequence of operation instances coupled with time bounds for their execution. The appendix contains examples of these concepts.

We are currently working with an extended version of Warren's WARPLAN algorithm [Wa,Ch,VF]. The algorithm adds second order logic features to Prolog's resolution-based inference machine. With our version it is possible to cope with goals consisting of conjunctions of positive and negative facts, which may be mutually interfering, and to associate time requirements with the goals. Moreover, our version takes future pre-recorded operations (see the previous section) into consideration and, to generate a plan, it essentially interpolates among them only those operation instances, if any, that are still needed to achieve the goal in hand.

Considering that to meet his goals the user may have either no plan at all, or a partial plan, or even a complete plan, the algorithm can either generate the plan or fill-up its gaps or simply check if the plan is valid (does not violate any constraint) and indeed achieves the goal (and, possibly, additional effects). In case of failure, the algorithm can identify the reason for the failure, so as to make it possible for the cooperative system to either offer to warn when removable obstacles no longer hold or to produce an alternative plan or to explain the failure to the user, if nothing else is possible.

The algorithm can also help match partial or complete plans against those kept in the repertoire of typical plans, mentioned before. This helps identify, at the earliest possible moment in a session with the user, what plan he may be trying to elaborate [AP,FM]. Whenever one or more alternative plans are feasible, besides that indicated by the user, one must decide whether the user's plan should be preferred or if the system should try to find an "optimal" or at least a "better" plan. The choice should consider that alternative plans may have different effects besides those originally intended and may differ with respect to time schedules.

6. SIMPLE EXAMPLES OF COOPERATIVE BEHAVIOUR

Consider a simple-minded academic database of a university department where the facts are that courses are offered with a certain number of credits and students take courses. Laboratory classes are a sub-class of courses ("lab is-a course"), having admittance

requirement as an additional attribute. There are operations to *offer* and to *cancel* a course, to *enroll* a student in a course and to *transfer* a student from a course to another. The pre-condition to the last two operations is that the course that the student will be taking is being offered. The pre-condition to cancel a course is that no student be taking it. The offer operation can only be executed in March. An integrity constraint forbids that, at any database state, courses C1 and C5 be simultaneously offered.

The users of the application are the department chairman, teachers and students. The chairman's profile records his personal policy that courses with fewer than five students should not be offered.

A few examples of how to handle requests in a cooperative way are given in the sequel. For expository reasons, they are expressed in English rather than in the original formal notation [FC]. The appendix contains the formal specification of this database.

R1 - "What is the admittance requirement for course L4?"
A rule of type *pre* can be applied to *correct* the query to "What is the admittance requirement for lab L4?". The rule recognizes the rather common error of referring to a more general entity class when an attribute that belongs to a more specialized class is involved.

R2 - "Is course C5 being offered?"
If R2 is asked by a student, a *pre* rule will *complement* the query to "Is course C5 being offered? with how many credits?", since the student is probably considering whether he should take the course and hence the number of credits assigned is relevant to his decision.

R3 - "How many students are taking course C1?"
If the question is posed by the chairman of the department and the result is less than five, an *s_post* rule will *complement* the query to also ask "To which courses (if any) can each student now taking C5 be transferred?". The rule assumes that the chairman may want to cancel C5, which is possible only after there are no students taking it. This is one of the many cases where plan generation is used.

R4 - "Which courses are being offered and with how many credits?"
Suppose the answer is "Course C2 with 2 credits, course C8 with 2 credits, course L4 with 2 credits". An *s_post* rule will *articulate* the answer changing it to "Courses C2, C8 and L4 are being offered, all with two credits". The answer becomes more informative, by stressing that the number of credits is the same. The rule makes use of most specific generalization [WM].

R5 - "Is course C5 being offered with 2 credits?"
Suppose the answer is negative and that the session's log registers that the same question about being offered with 2 credits was asked before by the user with respect to other courses. An *f_post* rule will generate an *alternative* query "Find some course being offered with 2 credits". This applies again most specific generalization, this time to find which element of the query can be turned into a variable (in place of the reference to *some* course).

R6 - "Is course C5 being offered?"
If the question is asked by a student and the answer is negative, an *f_post* rule will offer to *alert* him if and when the course is ever offered. The assumption, as before, is that if the student asks about a course he is planning to take it, a pre-condition being that the course be offered.

R7 - "offer C5 with 3 credits"
This is an update request that would normally be formulated by a teacher. Suppose it fails. An *f_post* rule will find the reason for the failure, to be *explained* to the user. It may be the case, for instance, that course C1 is already being offered, so that if C5 were offered the integrity constraint excluding their simultaneous offering would be violated.

A few remarks are in order. Several rules may be applicable to the same request, as might be the case of R2, R5 and R6. The rules used to process R1, R4 and R5 have a domain-independent scope. Time considerations provide ample opportunities for further manipulating the requests; for instance, alternative or additional reasons for the failure in R7 might be that the operation to offer C5 had already been pre-recorded in the database for a future date, and/or that the current attempt were made outside the month of March.

7. CONCLUSION AND FUTURE DIRECTIONS

To more concretely investigate the concepts discussed here, we are building a prototype, as part of project NICE. As in [CD], we are not concerned with natural language understanding, having instead made our option for a formal logic programming notation, expressing entity-relationship and abstract data types concepts. Entity and relationship instances have keys which establish their identity [PC] across the is-a links. A frame construct is provided to collect sets of attribute-values.

NICE is written in IBM Prolog [WM] and keeps the time-stamped records of actions in relational tables, using SQL/DS through an enhanced interface that allows a uniform

treatment of Prolog clauses and SQL tuples. Besides the plan and schedule generator and the SQL extended interface, several tools are being combined to form the NICE prototype. They support, among other features, the entity-relationship language layer, request modification, a query-the-user facility which communicates with the user in pseudo natural language, a user-monitor to handle the session logs and to invoke "demons", frame manipulation utilities.

The implemented entity-relationship language layer proved sufficient to express some useful domain-independent rules to handle cases discussed in [Ka] in the context of natural language. If a user's request refers to an attribute of an entity or relationship instance, the user clearly reveals his *presupposition* that the instance exists: then, if the request fails, "f_post" rules can check such presuppositions and warn the user if it they are not confirmed. Also, "pre" rules can correct the common mistake of asking about an attribute of an entity when the attribute really belongs to another related entity, sometimes involving the traversal of several relationship links. However, further additions to the language may still be considered, in particular *case* tags to qualify the participation of each entity class in a relationship; other domain-independent rules might then be derived from *conceptual dependencies* [Sc].

The available frame constructs seem adaptable to user-modelling. Information about a user coming from various sources can thus be combined, providing, among other items, personal data, role in the application domain (e.g. teacher, student, chairman) and even database facts referring to the user. Indeed, users are entities and user classes form is-a hierarchies with inheritance of attributes. On the other hand, the *stereotype* concept [Ri], whereby defaults are assigned to user classes, lends itself well to frame representation.

Future research will concentrate on enlarging the set of domain-independent rules provided and on investigating the feasibility of tools to help knowledge acquisition about the application domain, thereby assisting the formulation of the domain-specific rules. The contents and structure of user profiles [KW, NS] will also be examined as part of the problem of conciliating the user's and the system's views.

REFERENCES

[AP] J. F. Allen and C. R. Perrault - "Analyzing intentions in utterances" - Artificial Intelligence 15, 3 (1980) 143-178.

[BJ] L. Bolc and M. Jarke (eds.) - "Cooperative Interfaces to Information Systems" - Springer-Verlag (1986).

[CC] W. Chu, Q. Chen and R-C. Lee, "Cooperative Query Answering via Type Abstraction Hierarchy", Proc. Int. Working Conf. on Cooperating Knowledge based Systems, Univ. Keele, UK (Oct. 3-5, 1990).

[CD] F. Cuppens and R. Demolombe - "Cooperative answering: a methodology to provide intelligent access to databases" - Proc. of the Second International Conference on Expert Database Systems - L. Kerschberg (ed.) - Benjamin/Cummings (1989) 621-643.

[Ch] D. Chapman - "Planning for conjunctive goals" - Artificial Intelligence - 32, 3 (1987) 333-377.

[De] Deen, S.M., "Cooperating Agents - A Database Perspective", Proc. Int. Working Conf. on Cooperating Knowledge based Systems, Univ. Keele, UK (Oct. 3-5, 1990).

[FC] A. L. Furtado and M. A. Casanova - "Plan and schedule generation over temporal databases" - Proc. of the 9th International Conference on Entity-Relationship Approach, Lausanne, Switzerland (Oct. 8-10, 1990).

[FM] R. J. Firby and D. McDermott - "Representing and solving temporal planning problems" - in "The Knowledge Frontier" - N. Cercone and G. McCalla (eds.) - Springer-Verlag (1987) 353-413.

[Ka] S. J. Kaplan - "Cooperative responses from a portable natural language query system" - Artificial Intelligence, 19 (1982) 165-187.

[KS] R. Kowalski and M. Sergot - "A logic-based calculus of events" - New Generation Computing 4, 1 (1986).

[KW] A. Kobsa and W. Wahlster (eds.) - "User Models in Dialog Systems" - Springer-Verlag (1989).

[NS] E. J. Neuhold and M. Schrefl - "Dynamic derivation of personalized views" - Proc. of the 14th VLDB Conference (1988) 183-194.

[PC] K. Parsaye, M. Chignell, S. Khoshafian and H. Wong - "Intelligent Databases - Object-Oriented, Deductive Hypermedia Technologies" - John Wiley (1989).

[Ri] E. A. Rich - "User modeling via stereotypes" - Cognitive Science, 3 (1979) 329-354.

[Sc] R. C. Schank - "Conceptual information processing" - North-Holland (1984).

[St] M. R. Stonebraker - "Implementation of integrity constraints and views by query modification" - Proc. ACM SIGMOD International Conference on Management of Data (1975).

[TF] L. Tucherman and A. L. Furtado - "Update-oriented database structures" - Proc. of the Second International Conference on Expert Database Systems - L. Kerschberg (ed.) - Benjamin/Cummings (1989) 185-203.

[VF] P. A. S. Veloso and A. L. Furtado - "Towards simpler and yet complete formal specifications" - in "Information systems: theoretical and formal aspects" - A. Sernadas, J. Bubenko and A. Olive (eds.) - North-Holland (1985) 175-189.

[Wa] D. H. D. Warren - "WARPLAN: a system for generating plans" - memo 76 - University of Edinburgh (1974).

[We] B. L. Webber - "Questions, answers and responses: interacting with knowledge base systems" - in "On knowledge base management systems" - M. L. Brodie and J. Mylopoulos (eds.) - Springer (1986).

[WM] A. Walker, M. McCord, J. F. Sowa and W. G. Wilson - "Knowledge systems and Prolog" - Addison-Wesley Pub. Co. (1987).

APPENDIX - EXAMPLES

The academic database introduced in section 6 can be formally specified as follows:

```
% declaration of ER classes

entity(student,s_numb).
entity(course,c_numb).

relationship(takes,[student,course]).

attribute(course,credits).

domain(s_numb,X) <- on(X,[u,v]).
domain(c_numb,X) <- on(X,[c1,c2,c3]).
domain(credits,X) <- in_range(X,1,2).

% declaration of operations

added(course¢C,offer(C,N),*).
added(course¢C\credits(N),offer(C,N),*).
added(takes#[student¢S,course¢C],enroll(S,C),*).
added(takes#[student¢S,course¢D],transfer(S,C,D),*).

deleted(course¢C,cancel(C),*).
deleted(takes#[student¢S,course¢C],transfer(S,C,D),*).

precond(offer(C,N),true,*).
precond(cancel(C),X,St) <-
  neg_conj_at(X,takes#[student¢*,course¢C],St).
precond(enroll(S,C),course¢C,*).
precond(transfer(S,C,D),course¢D,*).

time_bounds(offer(C,N),[Y,3,1]:[Y,3,31],*) <-
  now([Y!*]).

% declaration of constraints

imposs(course¢c1@T & course¢c5@T).
```

As compared with section 3, we remark that the notation has been slightly extended so that, for example, the clauses for operations have an additional state parameter (the last parameter). The syntax used for clauses is in fact exactly that of IBM Prolog [WM], which allows the definition of additional operators (e.g. #).

The above specification implies the following. The first two added clauses imply that courses have an independent existence and that they are created by the offer operation; by contrast, the last two added clauses indicate that students exist only as long as they are taking a course (a case of weak entity).

The first deleted clause indicates that, after executing the cancel operation with parameter C, the fact course¢C ceases to hold. Likewise, the second deleted clause implies that, after executing the transfer operation with parameters S, C and D, the fact takes#[student¢S,course¢C] ceases to hold.

The first precond clause says that the offer operation can always be executed. The second precond clause implies that a course can only be cancelled after all students taking it at the state just before the operation is executed cease to take the course (the appropriate sub-goal, taking the form of a conjunction of negated facts, is generated by predicate neg_conj_at). The last two precond clauses imply that students cannot be enrolled in, or transferred to, a course unless the course exists;

The time_bounds clause imposes the temporal constraint that courses can only be created in March, and the imposs clause captures the constraint that courses c1 and c5 cannot be simultaneously offered.

Certain other constraints are also a implicit consequence of the above specification. For example, transferring a student from a course to the same course is disallowed by the productivity requirement, which also guarantees that credits be a single-valued attribute of courses.

Now suppose the current date is March 23/1990 and the following operation instances have been recorded (in the appropriate files), with the time stamps reduced to year/month/day for readability:

```
offer([90,3,1],c1,1).
enroll([90,3,2],u,c1).
offer([90,3,25],c2,1).
enroll([90,4,2],u,c2).
enroll([90,4,18],v,c2).
```

Clearly the last three operation instances belong to the future.

From the first two clauses and the added clauses, one may infer that, at the current state, course c1 is offered with one credit and student u takes it. From the five clauses and the

added clauses, one may infer that on April 18th all this is still true and, in addition, c2 is offered with one credit and it is taken by students u and v.

Some queries to retrieve facts like these are:

```
<- now(T) & course¢cl\credits(N) @ T.
<- takes#[student¢u,course¢C] @ [90,4,20].
```

As a result of the first query, T will be instantiated with the current time, read from the system's clock, and N will be instantiated with 1. The second query causes C to be instantiated with c1 and, if backtracking is requested, with c2.

The execution of the first command below will fail, since 90/4/6 exceeds the time bounds for offering courses (see the bime_bounds clause above), whereas the second will succeed and it will propagate to revoke both enroll operation instances involving course c2.

```
<- execute(offer[c3,2],[90,4,6]).
<- revoke(offer(c2,1)).
```

Finally, let us look at the generation of a plan with a schedule. Starting with the original records of operation instances (thus assuming that the above revoke command was not entered), consider the goal: "course c1 is not offered, a course with two credits is offered and student u takes it", with the requirement that "the plan to reach a state where all this holds should not begin before March 10th and should be completed by April 10th". The command to generate the corresponding plan P with schedule S is:

```
<- sch_plans(¬course¢cl & course¢X\credits(2) & takes#[student¢u,course¢X],
             [90,3,10]:[90,4,10], P, S).
```

with P and S being instantiated with sequences that are best displayed in tabular form. Notice that the creation of course c3 (with two credits) has been scheduled inside the period of time when this is permitted.

```
Plan                    Schedule
offer(c2,1) ---------   90/3/25
offer(c3,2) ---------   90/3/25:90/3/31
enroll(u,c2) --------   90/4/2
transfer(u,cl,c3) ---   90/4/2:90/4/10
cancel(cl) ----------   90/4/2:90/4/10
```

IBM Brasil

Rio Scientific Center
P.O. Box 4624
20.001 Rio de Janeiro - RJ
Brazil

Limited Distribution
Printed in Brazil
File: CCR0118T TEX/CCR0118C SCRIPT