

NICE: A COOPERATIVE ENVIRONMENT FOR THE USE OF INFORMATION SYSTEMS

Marco A. Casanova and Antonio L. Furtado¹

Centro Cientifico Rio
IBM Brasil
P.O. Box 4624
20.001, Rio de Janeiro, RJ - Brasil

(VNET: CASANOVA at RIOVMSC)

ABSTRACT

The purpose of project NICE is to investigate knowledge based methods and to develop software prototypes that provide a *cooperative environment* for the use of information systems. The paper 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.

Towards cooperative environments

The purpose of project NICE 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 [We,BJ] 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

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.

¹ also with the Departamento de Informatica, Pontificia Universidade Catolica do RJ.

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 behavior, the system cannot execute a request *literally*, but rather it has to modify the request appropriately. The *instruments* we use for this purpose are *rules*, classified as follows. First, we distinguish between *domain-independent* and *domain-dependent* rules, the latter being specific to the application domain on hand. Rules can also be classified according to the phase at which they are applicable:

- "pre" rules - preliminary to command execution
- "s_post" rules - following successful execution
- "f_post" rules - following execution in case of failure

Rules of the "pre" class can *correct* or *complement* a request. Complementing a request means doing more than has been asked. Sometimes it is possible to find out, before executing a request, that additional information (for queries or for updates) should be supplied, or additional changes should be effected (in case of updates).

Alternatively, the need to complement a request may be detected only after the execution of the request. A successful execution may trigger "s_post" rules that will generate and execute new requests. 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 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: pre-conditions that do not hold when the request is posed may 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 component of A is the transformation of R into a modified request R'. Condition C refers, of course, to the parameters of R, but it may also be based upon an environment 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.

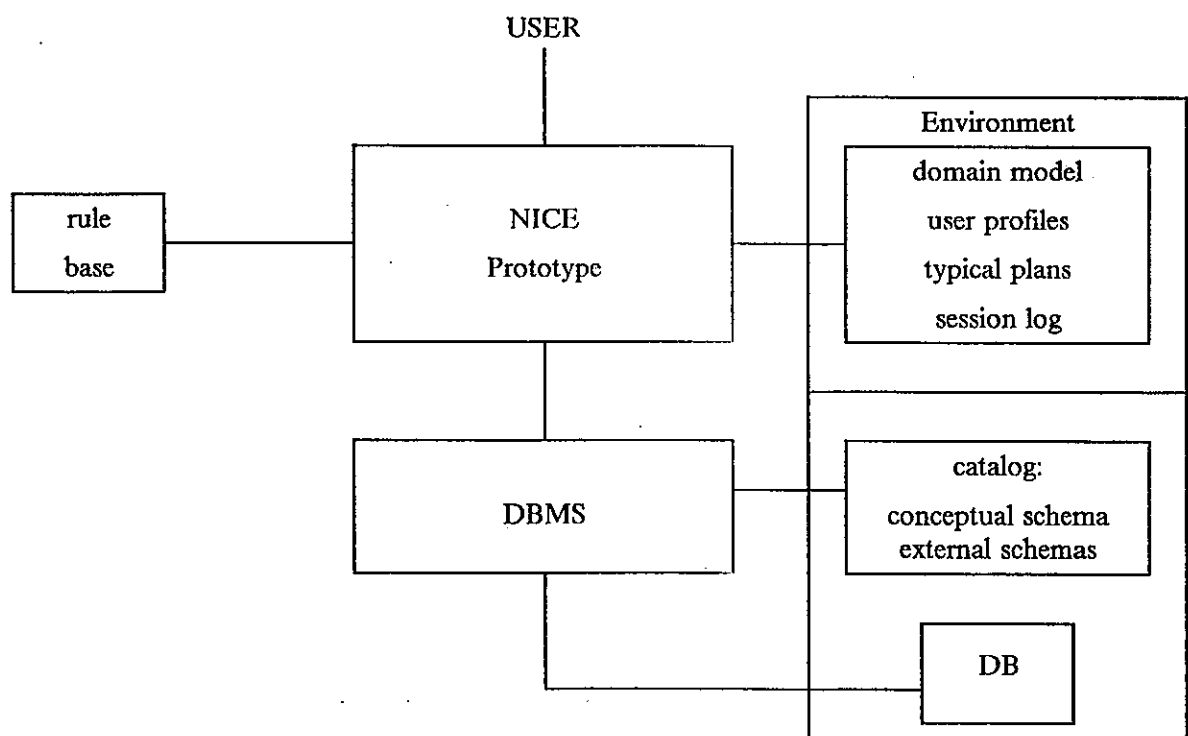


Figure 1: Architecture of the Prototype.

Clausal specification

We adopted a uniform way to specify the conceptual schema, the operations and the integrity constraints. The conceptual model is basically the entity-relationship model with is-a hierarchies. The conceptual schema of a database is specified through clauses that indicate which entities exist and how they are interrelated. A *database* for a conceptual schema is a set of facts (unit positive clauses), 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 database *state* at time t is 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.

Integrity constraints may also be declared, still using classes of clauses, without referring to specific operations. 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:



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 (see below). 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 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 operation, there may be other operations that achieve fact2, and have perhaps different pre-conditions and other additional effects

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 kept explicitly 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 in 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.

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.

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 on 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.

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 requires that, at no 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.

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.

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. Rules 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.

Project status and future directions

To more concretely investigate the concepts discussed here, we are building a prototype. 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 unification and frame generalization utilities.

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] 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).
- [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.
- [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.
- [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 (1990).
- [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).
- [PC] K. Parsaye, M. Chignell, S. Khoshafian and H. Wong - "Intelligent Databases - Object-Oriented, Deductive Hypermedia Technologies" - John Wiley (1989).
- [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).