FOLRE:
Towards a System for the Integrated Treatment of Updates and Rule Enforcement in Deductive Databases

T. Urpí, E. Teniente, J. A. Pastor, E. Mayol, C. Martín

Universitat Politècnica de Catalunya
Dept. de LSI - Facultat d'Informàtica
Pau Gargallo, 5
08028 Barcelona, Catalonia
{urpi | teniente | pastor | mayol | martin}@lsi.upc.es

ABSTRACT

We overview the architecture of a newly-born *deductive database management system* that is to provide advanced update and rule enforcement capabilities within an integrated and theoretically well-founded framework. The system builds up on previous theoretical results and systems of others -for the semantics, query, storage and transaction management needs- and of our own previous research and prototypes -for the advanced update and rule enforcement purposes. We hope that our approach will help to fill an important gap within current deductive database management systems prototypes and products.

KEYWORDS
Deductive Databases; Database Updating;
View, Integrity and Trigger Rule Enforcement;
Data Base Management Systems

October 1994

1. Introduction

The emerging needs for more "intelligence" within information systems and other computer-based applications are posing a high demand on infrastructure technology for their implementation and use. Within this market-pull context, Deductive Database Management Systems (DDBMS) seem to finally begin to evolve from research tools into practical database systems. The large amount of theoretical research devoted to the field of Logic or Deductive Databases during the past decade is now materializing in a few DDBMS implementations, either at the prototype or at the commercial product level [VLDB94].

Significantly though, while [VLDB94] presents deductive databases as the main effort to attack "one of the most fundamental uses of a computer, *i.e.* to store and retrieve information", the same important reference identifies as some of the principal theoretical and practical problems to be overcomed by current DDBMS those most closely related to the -also fundamental- task of updating information. Solving these problems is even more important in deductive databases than in previous models, since these databases characteristically offer view, integrity and trigger rules that must be enforced when confronting updates. Furtherly, such rules expand the basic updating possibilities of databases with more advanced options.

Certainly, most of the research -and resulting systems- have concentrated on providing theoretical foundations and efficient techniques for the storage and retrieval of large amounts of complex data. Deductive database semantics, (recursive) query processing and deductive storage systems are among the most popular themes within the area. On the contrary, just a few current DDBMS incorporate some mechanism for the -advanced- updating of data, other than updates to base relations. In this respect, these systems have not gone beyond what is already offered by their relational counterparts.

The above is true even when there are already satisfactory theoretical results for most of the deductive database update and rule enforcement problems, namely *view updating* and *materialization*, *integrity constraints checking* and *maintenance*, and *trigger condition monitoring*, among others. Several promising methods have appeared with the aim of solving each of those problems, some just at the theoretical stand while others further at a prototypical level. However, most of the methods have tackled their respective problem of interest in an isolated manner, without yet taking into account the strong interrelationships between all such problems.

Thus we arrive to the driving motivations of our previous research in deductive databases and the main goal of the FOLRE[1] project introduced in this paper: "The integrated treatment of

[1] **FOLRE** is both an accronym for "**F**acing **O**n towards **L**ogic database **R**ule **E**nforcement" and the name of the cover structure recently used again to tower higher up the typical Catalan human castles.

advanced update and rule enforcement problems in deductive databases through a specialized DDBMS".

For such purpose we build up on previous theoretical results and systems of others -for the semantics, query, storage and transaction management needs- and of our own previous research and prototypes -for the advanced update and rule enforcement purposes. Regarding our group's previous work on the subject, several competitive methods have been proposed -and some prototyped- for subjects such as integrity constraints checking and maintenance, view updating and materialization, and trigger condition monitoring. However, although developed on a common theoretical corpus, so far our own methods do share with previous ones (some of) the isolation critic mentioned above. To begin solving such shortcoming, and already within the premises of the FOLRE project, [TU95] has stated the theoretical basis for the specification and combination of the above problems within an integrated and well-founded framework.

Evolving from these theoretical results and heterogeneous prototype systems, we are starting the development of the FOLRE DDBMS. For this we are using a value-added layered architecture where a deductive front-end will take care of any advanced updating task -our job- while a pre-existing deductive back-end handles all storage, retrieval and transaction management issues.

This paper is organised as follows. After motivating the mission of our project with this introduction, section 2 reviews the basic concepts of deductive databases. Section 3 outlines the Data Definition and the Data Manipulation Languages of our system. Section 4 covers the overall architecture of the FOLRE system as well its linkages with our previous work. Finally, section 5 presents some conclusions and comments on implementation options.

2. Deductive databases

A deductive database D consists of three finite sets: a set F of facts, a set R of deductive rules, and a set I of integrity constraints. A fact is a ground atom. The set of facts is called the extensional database (EDB), and the set of deductive rules and integrity constraints is called the intensional database (IDB).

We assume that database predicates are either base or derived (views). A base predicate appears only in the extensional database and possibly in the body of deductive rules. A derived predicate appears only in the intensional database.

We also assume that database predicates (base or derived) can have two types of arguments: a vector of arguments k that form a key for the predicate and a vector of arguments x other than the key. We have then two types of predicates: those $P(k,x)$ with key and non-key arguments and those $P(k)$ with only key arguments. Moreover, we consider that the deductive database is allowed and stratified.

2.1 Deductive rules

A deductive rule is a formula of the form:

$$P \leftarrow L_1 \wedge \ldots \wedge L_n \quad \text{with } n \geq 1,$$

where P is an atom denoting the conclusion, and $L_1 \wedge \ldots \wedge L_n$ are literals representing conditions. Each L_i is either an atom or a negated atom. Any variables in $P, L_1, \ldots, L_n$ are assumed to be universally quantified over the whole formula. We also assume that the terms in the conclusion are distinct variables, and that the terms in the condition are either variables or constants.

2.2 Integrity constraints

An integrity constraint is a closed first order formula that the deductive database is required to satisfy. We deal with constraints that have the form of a denial:

$$\leftarrow L_1 \wedge \ldots \wedge L_n \quad \text{with } n \geq 1,$$

where each L_i, i = 1... n, is either an atom or a negated atom. Any variables in L_i are assumed to be universally quantified over the whole formula.

For the sake of uniformity, we associate to each integrity constraint an inconsistency predicate Icn, with or without terms, and thus they have the same form as the deductive rules. Then, we rewrite the former denial as Icn $\leftarrow L_1 \wedge \ldots \wedge L_n$.

3. Data Definition and Data Manipulation Languages

3.1 Data Definition Language

Our data definition language allows to express all the elements that may appear in a deductive database. Thus, the designer can define base predicates, derived predicates, integrity constraints and keys of base and derived predicates. Furthermore, in our system the designer may also specify some additional knowledge on these elements.

For derived predicates, the designer can specify if s/he wants to materialize them, that is to explicitly store its extension in the database. Moreover, s/he may also express the kind of changes (insertions, deletions and/or modifications) that the system must monitor on these predicates.

For integrity constraints the designer may distinguish between those constraints to be checked, so that any transaction that would violate them is to be simply rejected, and those to be maintained, which the system must try to compensate before are they violated by a transaction.

The designer may also define predefined transactions that are pre-compiled programs that correspond to the sets of updates that will often be required and, thus, that the system may optimize at compile time. In section 4, we will explain all these aspects in greater detail.

Example: Our example, is a database for an "Employment Office" that arranges labour interviews between its registered job applicants and employer companies. For the people administered by the office, it also keeps track of employees and, for legal reasons, of nationality status and of the existence of criminal records. Described in a preliminary version of our data definition language, the example is:

base predicate Emp (x: string)
  key: x
  description: x is an employee

base predicate App (x: string)
  key: x
  description: x is a job applicant

base predicate Comp (y: string)
  key: y
  description: y is an employer company

base predicate Int (x: string, y: string)
  key: x
  description: x has an interview with y

derived predicate
    Rr (x) ← Ra (x) ∧ ¬ Cr (x)
    Rr (x) ← Cit(x)
  key: x
  type: non-materialized
  description: x has right of residence if s/he is either a registered alien with no criminal record or a citizen

base predicate Cit (x: string)
  key: x
  description: x is a citizen

base predicate Ra (x: string)
  key: x
  description: x is a registered alien

base predicate Cr (x: string)
  key: x
  description: x has criminal records

base predicate Leg_ent (x: string)
  key: x
  description: x is a legal entity

derived predicate
    Cand (x) ← Int (x,y) ∧ Comp (y)
  key: x
  type: materialized
  monitor: deletions
  display x has been deleted as candidate
  description: x is considered a job candidate when s/he has an interview with an employer

integrity constraint
ic1 (x) $\leftarrow$ Emp (x) $\wedge$ $\neg$ Rr (x)
type: maintenance
description: Employees must be legal residents

integrity constraint
ic2 (x) $\leftarrow$ Emp (x) $\wedge$ App (x)
type: checking
description: Employees cannot be applicants

integrity constraint
ic3 (x) $\leftarrow$ Cand (x) $\wedge$ $\neg$ App (x)
type: maintenance
description: Candidates must be applicants

integrity constraint
ic4 (x) $\leftarrow$ Leg_ent (x) $\wedge$ $\neg$ Comp (x)
type: checking
description: Legal entities must be companies

predefined transaction Enter-employee (Per)
is {a set of updates specified with the Data Manipulation Language with formal parameters}

3.2 Data Manipulation Language

In our system, the user can request insertions, deletions and modifications of base and derived facts. These updates are specified in the following way:

ιP(**K**,**X**): denotes an insertion of the fact P(**K**,**X**), where **K** and **X** are vectors of constants.

δP(**K**,**X**): denotes a deletion of the fact P(**K**,**X**), where **K** and **X** are vectors of constants.

μP(**K**,**X**,**Y**): denotes a modification of the fact P(**K**,**X**) to P(**K**,**Y**), where **K**, **X** and **Y** are vectors of constants and **X** ≠ **Y**.

The user can also request updates that consist of a set of insertions and/or deletions and/or modifications of base and derived facts. For a more formal definition of our data manipulation language see [UO92].

We require the update requests to be effective. For instance, if the user requests the insertion of the fact Cand(Maria), the fact Cand(Maria) must hold in the current database.

Example1: An update request consisting of the deletion of derived fact Rr(Joan) together with the deletion of the base fact App(Maria) would be expressed in our language as { δRr(Joan), δApp(Maria) }.

Example2: The predefined transaction request explained in section 3.1 is:

predefined transaction Enter_employee (Per)
is { ιEmp(Per) }

Note that, for predefined transactions, the vector **X** refers to formal parameters that will be instantiated at execution time.

4. Architecture of the FOLRE system

Since the main emphasis of our system is on updating a deductive database, the system that we propose is composed of two parts. First, a front-end that acts as an interface with the user or the designer of the database and whose main goal is that of handling all the issues related with defining and updating a deductive database. Second, a deductive back-end platform that should include support for disk storage and full query facilities and transaction management. This idea is illustrated in figure 1.

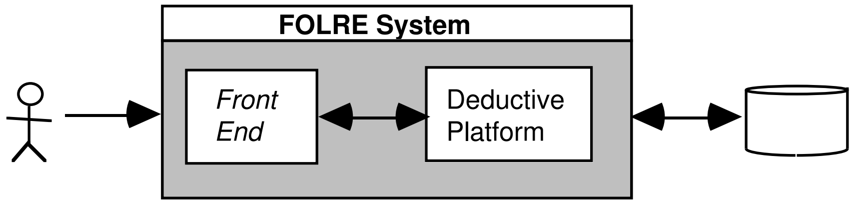

Figure 1 - System Architecture

We would like to point out that we propose a system which is independent of the deductive platform. There are several deductive platforms that satisfy our requirements (see [VLDB94] for a state-of-the-art report). For the full implementation of our system we have selected ECLiPSe [ECL94], mainly because we already had some experience using Megalog [Boc91], a precursor of ECLiPSe. One of the objectives of our project is to build the front-end in such a way that it could be easily ported to other deductive platforms fulfilling our requirements.

The front-end is divided into three components, as shown in figure 2: a Database Definition Manager, a Query Manager and an Update Manager. These components are explained in detail in the rest of this section.

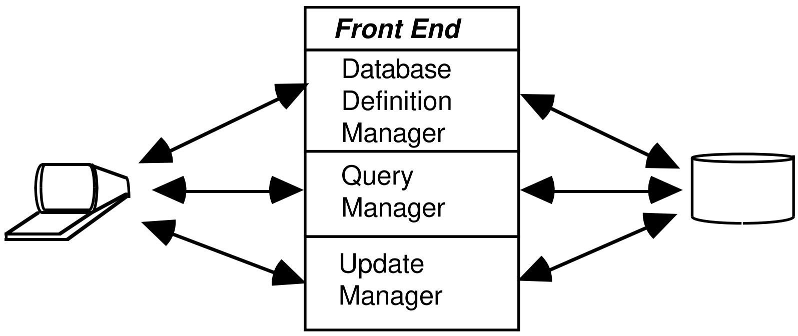

Figure 2 - Front-end Architecture

4.1 Database Definition Manager

The Database Definition Manager allows the database designer to specify all the aspects of a deductive database. In our system, the designer can specify base predicates, view predicates, integrity constraints, key constraints for base and view predicates, conditions to monitor and predefined transaction requests. The designer must also provide information about the views to be maintained, and s/he must distinguish between those constraints to be checked and those to be maintained. The components of the database definition manager are shown in figure 3.

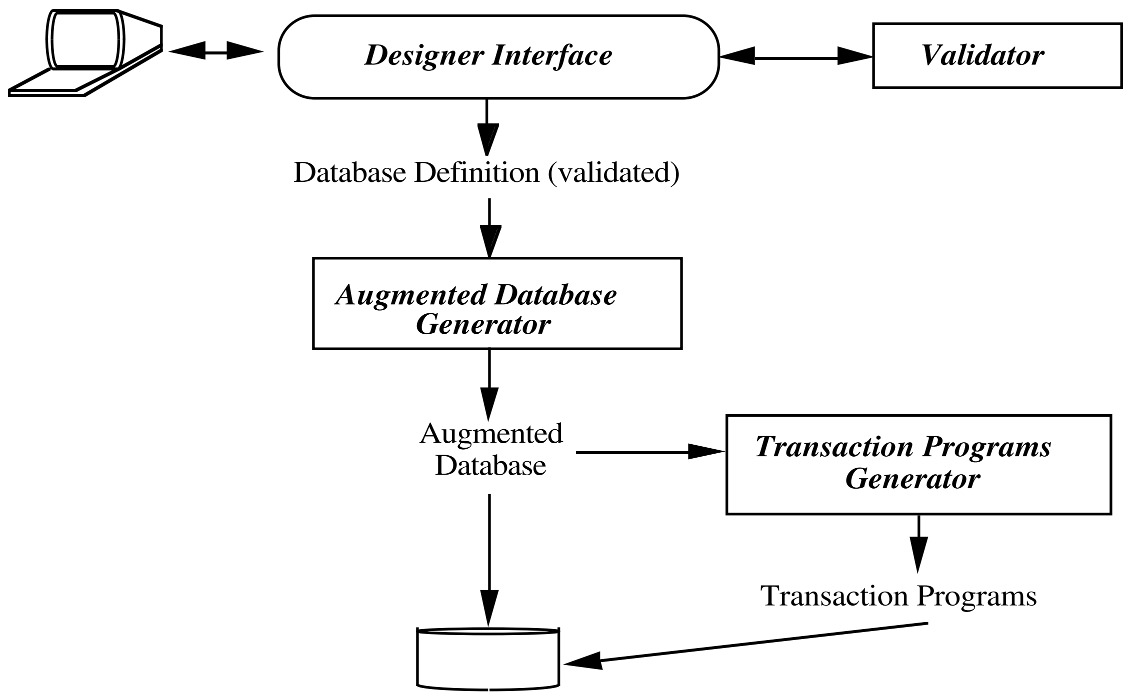

Figure 3 - Database Definition Manager Architecture

The Designer Interface handles all input/output interaction and allows the designer to specify all the deductive database aspects stated above. This specification is performed in the Data Definition Language described in the previous section.

The Validator is a tool offered to the designer to validate a deductive database definition. It allows to verify satisfiability of deductive rules, integrity constraints and conditions to monitor. Our previous work in this field relies on a prototyped method in the related context of deductive conceptuals models validation [CO92] and on the ideas outlined in [TU95] on how to use view updating techniques for dealing with these issues.

Given a validated deductive database definition, the Augmented Database Generator automatically derives the corresponding augmented database. That is, it obtains a model that specifies the changes that occur in a transition from and old state of a database to a new, updated state, as described in previous work of our group [Oli91, UO92, UO94]. The augmented database plays a key role in our system since all the methods that we have proposed for dealing with database updating problems are based on it. At the present stage, we have a prototype of this module implemented in QUINTUS Prolog.

In our system, the designer may also specify at definition time predefined transaction requests which consist of a set of proposed updates of base and view facts, defined with formal parameters. The Transaction Programs Generator automatically derives, at definition time, the corresponding transaction programs by taking into account, under the guidance of special control instructions, the view updates, the integrity constraints to be maintained and to be checked, the materialized views and the conditions to monitor. These transaction programs are physically stored in the database so that, at execution time, the users can instantiate the formal parameters with actual parameters and the updates given by the execution of the transaction programs are applied to the database, as we will explain in section 2.3. Presently, we have a prototype of this module implemented in LPA Prolog that handles view updating and integrity constraints checking and maintenance with insertions and deletions only [Pas94, PO94].

Example: as a comprehensive example, we will next show and comment on the resulting program automatically generated from our database example and the transaction request [Enter-employee(Per)]. With this request the designer wants a consistency-preserving transaction such that after executed for a concrete person, to be provided in parameter 'Per', we can guaranty that s/he is an employee; in other words, that inserts the person as employee, if necessary. This base update request will call, on its way, for a further view update request [ιRr(Per)], when performing integrity maintenance.

From the above request [Enter-employee(Per)] and the example database schema, our prototype generates the corresponding update transaction program code, contained in figure 4. Due to implementation reasons, a slightly different syntax is used for the various predicate types. In particular, base and derived predicates must begin with a lower-case letter, and meta-level update operators 'ι' and 'δ' are handled as prefixes 'i_' and 'd_', respectively. Horizontal and vertical lines have been added for ease of reading.

Within figure 4, line 1 controls if the person to be employed is already an employee, in which case line 2 exits the transaction without any updating. Otherwise if the person under consideration is not an employee, line 4 proposes to insert him/her as such. However, in this case, two of our integrity constraints are affected by such base update: Ic1, which has to be maintained with the appropriate compensating code, and Ic2, that must only be checked.

```
   trek_text([i_emp(Per)],
          '
1  ------------ if emp(Per) then
2  -------------|--- i_exit
3  -------------|- else
4  -------------|--- i_emp(Per) ,
5  -------------|----- if not rr(Per) then
6  -------------|------|- { i_rr(Per) }
7  -------------|------|--- either
8  -------------|------|----|-- if ra(Per) then
9  -------------|------|----|---|--- d_cr(Per)
10 -------------|------|----|---|- else
11 -------------|------|----|---|--- if not cr(Per) then
12 -------------|------|----|---|----|--- i_ra(Per)
13 -------------|------|----|---|----|- else
14 -------------|------|----|---|----|--- i_ra(Per) ,
15 -------------|------|----|---|----|--- d_cr(Per)
16 -------------|------|----|---|--- end_if
17 -------------|------|----|-- end_if
18 -------------|------|----|- or
19 -------------|------|----|-- i_cit(Per)
20 -------------|------|--- end_either
21 -------------|----- end_if ,
22 -------------|----- if app(Per) then
23 -------------|------|- i_abort
24 -------------|----- end_if
25 ------------ end_if
          '
         ). % end of trek text
```

Figure 4 - Transaction Program Code Example

With regard to the maintenance of Ic1, the generated code notices that, if we want to insert someone as employee (line 4) without a right of residence status (line 5), then we must grant right of residence to the person (lines 6 to 20) in order not to violate database consistency. For the granting of right of residence, we initially draw the proposal that ιRr(Per) should be pursued, which is shown in line 6 as a commented action preceding its further development. Later on, our method translates such view-update request into the needed base update instructions for granting right of residence (lines 7 to 20).

In fact, lines 5 to 21 would be the main body of the transaction synthesised from the [ιRr(Per)] request. Observe that for this view-update transaction, there are also two alternatives, corresponding to the two schema definitions for Rr(x). That is, we can give the right of residence to a person either by making him a non-criminal registered alien (lines 8 to 17) or by granting citizenship to such individual (line 19).

Regarding the code generated for checking Ic2, it states that, if we want to insert as employee (line 4) some applicant (line 22), then database consistency is to be preserved in this case by aborting the transaction (line 23).

The previous example shows how we address in an integrative way most of the updating problems presented in the introduction of this paper, namely view updating and integrity constraint checking and maintenance. For details on formal procedures and more examples, see [Pas94, PO94]. We plan to extend this compilation method and its implementation to handle the rest of the stated problems (view materializacion and condition monitoring) while also using the modification operator.

4.2 Query Manager

There is a large research effort on the field of query processing (see for example [BR86]). There are many systems that implement the most recent and efficient techniques and, thus, we can take advantage of these systems by using one of them as the selected platform. Therefore, in our system the Query Manager needs just to translate the query performed to the language of the ECLiPSe platform.

4.3. Update Manager

Given the main goal of our system, the Update Manager becomes the most important component of our system. The main purpose of this component is to allow the user to successfully update base and view facts. To this end, this component has to control integrity constraints, views to materialize and conditions to monitor.

Two kinds of updates are allowed by the system: on-line updates and predefined updates. Since these updates must be processed in a different way, in the architecture of the system we have distinguished between two different components: the On-line Update Manager and the Predefined Update Manager.

4.3.1. On-line Update Manager

The On-line Update Manager is the component of the system that permits the user to request at run-time a set of updates to be applied to the deductive database.

Given a user update request, the On-line Update Manager must change the contents of the deductive database such that the update is effectively performed, the integrity constraints remain satisfied, views to materialize are correctly updated and all conditions to monitor are inspected.

Figure 5 shows the architecture of the On-line Update Manager. It is composed by three modules: the View Updating and Integrity Constraints Maintenance Module, the Integrity

Constraints Checking Module and the Materialized View Maintenance and Condition Monitoring Module.

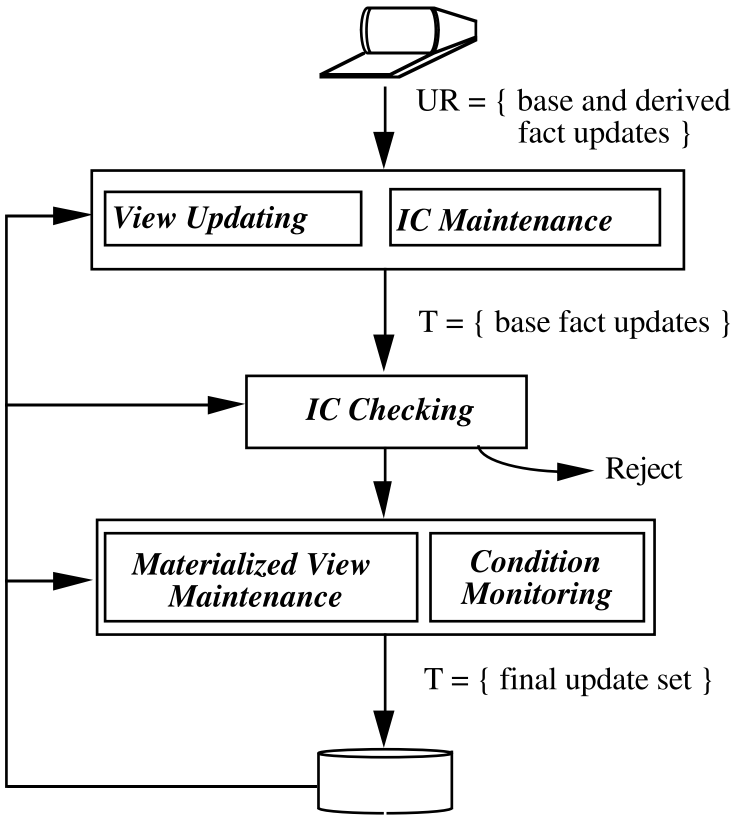

Figure 5- On-line Update Manager Architecture

In the following sections we will describe each module and we will illustrate them by means of an example.

4.3.1.1 View Updating and Integrity Constraints Maintenance Module

The View Updating and Integrity Constraints Maintenance module deals with two different but related problems: View Updating and Integrity Constraints Maintenance.

The view updating problem is concerned with determining how a request to update a view fact should be appropriately translated into updates of the underlying base facts. In general, several translations that satisfy a view update may exist. Much research has been devoted to this problem within the field of deductive databases; see for example [Dec90, KM90, GL91, AT92].

On the other hand, integrity constraints maintenance [ML91, Wüt93] consists of, given a consistent database state and a set of updates that violates some integrity constraint, determine a set of additional base fact updates to be appended to the original update set such that the resulting set satisfies all integrity constraints.

In our system, these two problems are handled as a whole as proposed in [TO92, MT93, TO95]. Given a user update request which consists of a set of base and view updates, the View Updating and Integrity Constraints Maintenance Module automatically translates the request into transactions consisting only of base fact updates. This translation is performed by taking into account the view updates and the subset of integrity constraints that the designer had specified to be maintained. Since, in general, several resulting transactions can be obtained the user may also guide the translation process and eventually choose between the resulting solutions. The theoretical results described in the papers mentioned above have been prototyped using POP Prolog and QUINTUS Prolog.

Example: Consider the database defined in section 2 and assume that the current contents of its extensional part is the following:

Cit (Joan)
App (Maria)
Int (Maria, UPC)
Cand (Maria)
Comp (UPC)
Legal_ent (UPC)

Consider also that the user requests the on-line update: UR = { δRr(Joan), δApp(Maria) }. That is, s/he wants to delete the fact that Joan has right of residence and the fact that Maria is a job applicant. The View Updating and Integrity Constraints Maintenance Module translates this update request into several sets of base fact updates that satisfy the view update δRr(Joan) plus the base update δApp(Maria) and at the same time maintain the constraints ic1 and ic3. In this case, three possible translations would be obtained:

T_1 = { δCit(Joan), δApp(Maria), δInt(Maria,UPC)}
T_2 = { δCit(Joan), δApp(Maria), δComp(UPC)}
T_3 = { δCit(Joan), δApp(Maria), μInt(Maria,UPC,UAB)}

The first event of each translation, δCit(Joan), corresponds to the translation of the view update δRr(Joan). That is, to delete the fact that Joan has right of residence, he must be deleted as registered alien. The second event of each transaction, δApp(Maria), corresponds to the deletion of the fact that Maria is a job applicant, explicitly requested by the user. Finally, the third event in T_1, T_2 and T_3 (δInt(Maria,UPC), δComp(UPC) and μInt(Maria,UPC,UAB), respectively) corresponds to the three possible ways of maintaining the integrity constraint ic3, which otherwise would be violated by the deletion of the fact that Maria is an applicant.

As we said before, the user may guide the translation process. In this example, we assume that for some particular reason the user discards T_1. Therefore, only T_2 and T_3 will be considered by the following modules.

4.3.1.2 Integrity Constraints Checking Module

Once we have transactions in terms of base fact updates, we have to verify that these transactions do not violate those integrity constraints that the designer had specified to be checked. This problem is known as integrity constraints checking [BDM88, GCM+94]. Before a transaction violates some of these integrity constraints, it must be rejected. Otherwise, it is accepted and our system proceeds with determining views to materialize and conditions to monitor.

Our contributions in this area propose to use the concept of augmented database for checking whether a transaction violates some integrity constraint [Oli91, UO92, UO94]. At the present stage, we have a prototype of this module implemented in QUINTUS Prolog.

Example: continuing our example, the Integrity Constraints Checking Module verifies whether T_2 and T_3 satisfy ic1 and ic4. In this case, the translation T_2 violates ic4 because, if it was applied, UPC would be a legal entity but not a company. On the other hand, the translation T_3 satisfies both constraints. Therefore, T_2 is rejected and the process continues with the only translation T_3.

4.3.1.3 Materialized View Maintenance and Condition Monitoring Module

A view is materialized by explicitly storing its extension in the database. Given a transaction, the materialized view maintenance problem consists in incrementally determining which changes must be performed to update accordingly the materialized views [CW91, GMS93].

As stated before, the database designer may also specify general conditions to be monitored. Thus, condition monitoring refers to the problem of incrementally monitoring the changes on a condition induced by a transaction [RCB+89, QW91].

We have also proposed a method for solving these two problems [UO92]. This method uses the augmented database to determine the updates to be performed in the materialized views.

Example: the Materialized View Maintenance and Condition Monitoring Module determines the changes induced to the derived predicate Cand due to $T_3 = \{\delta Cit(Joan), \delta App(Maria), \mu Int(Maria,UPC,UAB)\}$. In this example, this module would detect that the only induced change is δCand(Maria), which is due to the event fact μInt(Maria,UPC,UAB). That is, the modification of the fact that Maria has an interview with the company UPC by the fact that

Maria has an interview with the UAB, induces the deletion of Maria as a job candidate. Therefore, Cand(Maria) must be deleted from the materialized view and since deletion of Cand must be monitored, the associated text should be displayed.

Once our system has determined the views to materialize and the conditions to monitor, the overall resulting transactions are obtained. At this point, the user must choose the concrete transaction that must be physically applied to the database.

Example: we have obtained only one resulting transaction T_3 = {δCit(Joan), δApp(Maria), μInt(Maria,UPC,UAB), δCand(Maria)}. Therefore, to satisfy the original on-line update request UR = { δRr(Joan), δApp(Maria) }, the following actions should be performed:

- δCit(Joan): the base fact Cit(Joan) must be deleted
- δApp(Maria): the base fact App(Maria) must be deleted
- μInt(Maria,UPC,UAB): base fact Int(Maria,UPC) must be replaced by Int(Maria,UAB)
- δCand(Maria): the materialized fact Cand(Maria) must be deleted
the message "Maria has been deleted as a candidate" is displayed

4.3.2. Predefined Update Manager

Predefined updates correspond to the predefined transaction requests specified by the database designer. The Predefined Update Manager is the component of the system that allows the user to apply predefined transaction requests by executing their corresponding transaction programs. The Predefined Update Manager is composed by only one module: the Transaction Processor, as shown in figure 6.

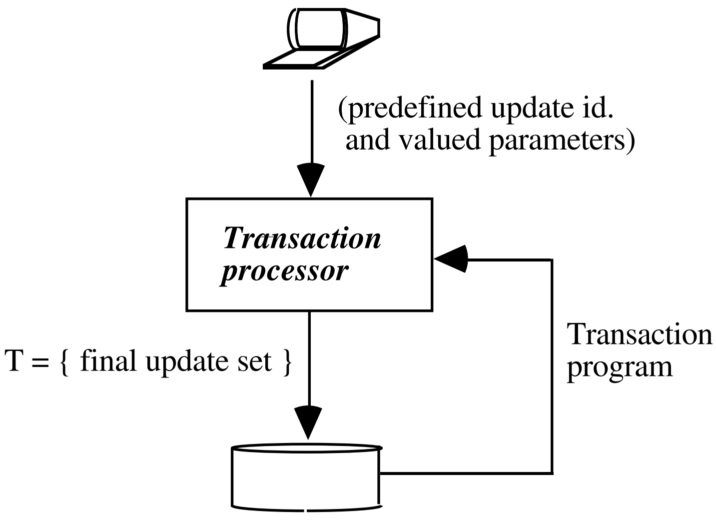

Figure 6 - Predefined Update Manager Architecture

To update a deductive database using a predefined parameterized transaction, the user must select the corresponding predefined transaction request and provide actual values to its parameters. The Transaction Processor uses these values to call the transaction program corresponding to the predefined transaction selected by the user. Then, it executes the transaction program and generates a set of ground updates to be applied to the deductive database.

Besides entering values for parameters, some further intervention will usually be expected from the user at run-time. For example, if the transaction in figure 4 (section 4.1) were provided to him/her as it is, then s/he would hold all choice responsibly when confronted with the 'either' control instruction. However, there could be other alternative choice strategies, such as leaving some of these decisions for the transaction processing system to take. It could use priority criteria either provided by the transaction designer from application domain semantics or inferred from extensional database statistics, or a combination of both. So far, this is an interesting open problem that we have not addressed yet.

With regard to assumed run-time environment, we consider delayed-update semantics for transaction-processing-time. That is, conditions within transaction programs always refer to the old database state, while proposed base updates are to be collected and finally committed as a whole and all at once to the database. Thus, there is no need to keep and/or to query any intermediate state.

5. Conclusions and further work

In this paper, we have described the architecture of the FOLRE system. The main goal of this system is to extend the current deductive database systems to provide more advanced updating capabilities like view updating, integrity constraints checking, integrity constraints maintenance, materialized view maintenance and condition monitoring.

At the present time, we have the main modules of our system protyped as separate units using various Prologs. The next step is to integrate all these modules on top of a common deductive platform, ECLiPSe, and to implement the rest of the modules. Thus, we would have a first version of the entire system. In a second stage, we plan to take into account efficiency issues which will probably imply the redesign of some of the current components of the system.

Acknowledgements

We would like to thank D.Costal, C.Quer, M.R. Sancho, J.Sistac and A.Olivé for many useful comments and discussions. This paper has been partially supported by the PRONTIC program project TIC94-0512.

References

[AT92] Atzeni, P.; Torlone, R. "Updating Intensional Predicates in Datalog", Data & Knowledge Engineering (8), 1992, pp. 1-17.

[Boc91] Bocca, J. "MegaLog, a platform for developing Knowledge Base Management Systems", International Symposium on Database Systems for Advanced Applications, Tokyo, 1991, pp. 374-380.

[BDM88] Bry, F.; Decker, H.; Manthey, R. "A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases", Proc of Extending Database Technology, Venice, 1988, pp. 488-505.

[BR86] Bancilhon, F; Ramakrishnan, R. "An Amateur's Introduction to Recursive Query Processing Strategies". Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington D.C., 1986, pp. 16-152.

[CO92] Costal, D.; Olivé, A. "A Model for Reasoning About Deductive Conceptual Models of Information Systems", Proc. of the 4th. CAiSE, Manchester, UK, 1992, pp 589-611.

[CW91] Ceri, S.; Widom, J. "Deriving production rules for incremental view maintenance", Proc. of the 17th. VLDB Conf., Barcelona, 1991, pp 577-589.

[Dec90] Decker, H. "Drawing Updates from Derivations", Proc. of the 3rd Int. Conf. on Database Theory (ICDT), Paris, 1990, pp. 437-451.

[ECL94] "ECLiPSe User Manual", Technical Report, ECRC, 1994.

[GCM+94] García, C; Celma, M.; Mota, L.; Decker, H. "Comparing and Synthesizing Integrity Checking Methods for Deductive Databases", Proc. of the 10th ICDE, Houston, USA, 1994, pp. 214-222.

[GL91] Guessoum, A.; Lloyd, J.W. "Updating Knowledge Bases II", New Generation Computing, Vol. 10, 1991, pp. 73-100.

[GMS93] Gupta, A; Mumick, I.S.; Subrahmanian, V.S. "Maintaining Views Incrementally", ACM SIGMOD, Washington, 1993, pp. 157-166.

[KM90] Kakas, A.; Mancarella, P. "Database Updates through Abduction", Proc. of the 13th Int. Conf. on Very Large Data Bases (VLDB), Brighton, 1987, pp. 61-70.

[ML91] Moerkotte, G; Lockemann, P.C. "Reactive Consistency Control in Deductive Databases", ACM Transactions on Database Systems, Vol. 16, No. 4, December 1991, pp. 670-702.

[MT93] Mayol, E.; Teniente, E. "Incorporating Modification Requests in Updating Consistent Knowledge Bases", Fourth International Workshop on the Deductive Approach to Information Systems and Databases, Lloret, 1993, pp. 335-360.

[Oli91] Olivé, A. "Integrity Checking in Deductive Databases", Proc. 17th Int. Conf. on Very Large Data Bases (VLDB), Barcelona, 1991, pp. 513-523.

[Pas94] Pastor, J.A. "Extending the Synthesis of Update Transaction Programs to handle Existential Rules in Deductive Databases", 5th Int. Workshop on the Deductive Approach to Information Systems and Databases, Aiguablava, Catalonia, 1994.

[PO94] Pastor, J.A.; Olivé,A. "An Approach to the Synthesis of Update Transactions in Deductive Databases", to appear in Proc. of the CISMOD'94 (Fifth International Conference on Information Systems and Management of Data), Madras, India, October 1994.

[QW91] Qian, X.; Wiederhold, G. "Incremental recomputation of active relational expressions", IEEE Trans. on knowledge and data engineering, Vol. 3, No. 3, September 1991, pp. 337-341.

[RCB+89] Rosenthal, A.; Chakravarthy,S.; Blaustein, B.; Blakeley, J. "Situation monitoring for active databases", Proc. of the 15th VLDB Conf., Amsterdam, 1989, pp. 455-464.

[TO92] Teniente, E.; Olivé. A. "The Events Method for View Updating in Deductive Databases", Proc. Int. Conf. on Extending Database Technology (EDBT'92), Vienna, 1992, pp. 245-260.

[TO95] Teniente, E.; Olivé. A. "Updating Knowledge Bases while Maintaining their Consistency", to appear in the VLDB Journal, 1995.

[TU95] Teniente, E.; Urpí, T. "A Common Framework for Classifying and Specifying Deductive Database Updating Problems", to appear in the proceedings of the 11th ICDE, Taiwan, 1995.

[UO92] Urpí, T.; Olivé, A. “A Method for Change Computation in Deductive Databases”, Proc. of the 18th Int. Conf. on Very Large Data Bases (VLDB), Vancouver, 1992, pp. 225-237.

[UO94] Urpí, T.; Olivé, A. "Semantic Change Computation Optimization in Active Databases", Proc. of the 4th Int. Workshop on Research Issues on Data Engineering- Active Database Systems (RIDE -ADS'94), Houston, USA, 1994, pp. 19-27.

[VLDB94] The VLDB Journal "Special issue on prototypes of deductive database systems". Vol. 3, Num. 2, April 1994.

[Wüt93] Wüthrich, B. "On Updates and Inconsistency Repairing in Knowledge Bases", Int. Conf. on Data Engineering, Vienna, 1993, pp. 608 - 615.