VI-SDB: A Convivial Approach for Description and Manipulation of Deductive and Stratified Databases

Amel Touzi
Faculty of Sciences of Tunis, Tunisia University, Tunisia

Abstract: *Although deductive databases is based on a well established formalism, they didn't know the expected success. Their use was limited to the academic purpose. Indeed, the deductive database management systems are judged abstract, rare in commercial offers, and often expensive. In among the abstract concepts of the deductive databases, we mention the case of the negation and its treatment by the stratification. In this paper, we propose a convivial approach that aims to make transparent theses concepts relatively abstracted and to permit a friendly usse of deductive databases and stratified database. This approach permits to simplify concepts, which always remain delicate for this type of databases users or designers, like (1) the definition of a deductive and/or stratified database (2) the study of the stratifiability, (3) the determination of the maximal stratification, (4) the incremental definition of strata and (5) the checking of integrity constraints. These operations become more delicate if the database is voluminous. The proposed system supports rules update and is not limited to facts updating as in known deductive systems. This approach is implemented and validated with VI_SDB tool based on an extension of predicates nets.*

Keywords: *Deductive database, stratified database, EPN formalism, GUI.*

Received March 18, 2007; accepted May 30, 2007

1. Introduction

The integration of logic into Relational DataBases (RDB) has begun since the 1970's [6, 17, 20]. The main objective of this integration is to enrich DataBases Management Systems (DBMS) with deduction capacity, which is very useful for many applications such as medicine, cartography and expert systems. These systems are known as Deductive DataBases (DDB) [6]. Several prototypes of Deductive Databases Management Systems (DDBMS) have been proposed in literature [18, 19] as EKS, LOLA, CORAL, LDL and LDL++. Some of them have been marketed like EKS.

Even though DDB are based on a well established formalism [5, 17], which is the first-order logic [2, 15], they didn't meet the expected success. The DDBMS was mainly used for academic purposes [16, 19]. Indeed, the DDBMS are judged abstract, absent from commercial offers, and expensive. The reasons of these problems are:

- The lack, or even the absence, of uniform language standardization for the definition and manipulation of the database.
- The absence of graphic interfaces simplifying the concepts of DDB and SDB, as those which exist for RDB. As example, we note the access DBMS which presents a simple and convivial interface allowing to a non-specialist user to define and to handle RDB.

To palliate this problem, several languages of production rules type have been proposed [6, 9, 13, 14, 16, 17, 19, 20] as RDL, DLP, RDL/C and DATALOG. All these languages, including the standard DATALOG, suffer from several important limitations notably the limitation to the UPDating (UPD) of facts and the absence of a homogeneous and uniform framework to handle useful concepts to any application.

In this paper, we propose a convivial approach which proposes to make the DDBMS non abstract and more clear for the users. Indeed, the concepts of DDB, SDB, stratifiability of a SDB, maximal stratification of a SDB, incremental definition the SDB strata and checking of Integrity Constraints (IC) in the SDB remain always delicate for the user or the designer of the DDB and SDB. These operations become more delicate if the database is very large. To validate our approach, we implemented a tool, VI_SDB, based on the concept of the Stratified Extended Predicates Nets (SEPN). SEPN is an Extension of Predicates Nets. We already used these Nets for modelling and management of these databases [3, 7, 8, 11, 12, 20]. Applied to DDB and/or SDB these nets offer a convenient and expressive environment for modelling deduction techniques. VI_SDB allows the user to define and handle its database in two different manners, either

textually using an editor, or graphically by handling SEPN.

The rest of the paper is organized as follows. In section 2, we give the basic concepts of DDB and SDB. In section 3, we study the effect of update operations on the SDB. Section 4 describes our approach based on SEPN. The correspondence between SDB and SEPN is presented in section 5. Section 6 presents VI_SDB tool. Finally in section 7, we conclude and present future work.

2. Deductive Databases and Stratified Databases

DDB result from the intersection of Artificial Intelligence techniques and the databases. Nowadays, the most used model for databases manipulation is the Relational model. However, the new objectives target the knowledge rather then the data manipulation. It is about systems that stock the information in a manner suitable to automatic inference, and that recover them by applying inference mechanism to the registered information. The DDB is defined as a set of explicit facts, making Extensional DataBase (EDB) and a set of deduction rules allowing deduction of new facts, making the Intentional DataBase (IDB) [6].

A DDB is modelled by a definite program (a set of Horn clauses) [2, 15, 17, 21]. However, the definite programs inevitably pose the problem of limitation of their expressivity. Indeed, in most real world situations, we need to express negative information, and so the notion of normal programs was introduced. However, we are facing the problem of determination of a canonical semantics for these programs in a unique way [1, 6]. The stratified programs represent an efficient solution to this problem. The basic principle is the subdivision of the logic program in strata, such as a literal negative that cannot be used in the body of a rule unless it has already been defined in the rules that precede it [1, 6, 15, 10].

In the following, we present basic definitions related to stratified programs [1, 6, 15, 10]. Let P be a logic program. A predicate symbol q definition is the set of all the clauses of the program P having q at the head of the clause. The dependency graph of a program P (Dp) is composed by a set of nodes connected by arcs. Each node represents a predicate of P. The arc matching r and q, noted (r, q), belongs to Dp if there is a clause in P using r in its head and q in its body. We say r refers to q. If a predicate q appears positively (respectively, negatively) in the body then (r, q) is called positive (respectively, negative) arc.

A program is stratifiable if and only if its dependency graph does not contain any circuit containing a negative arc.

A program P is stratify if there is a partition $P= S_1 \cup ... \cup S_n$, called stratification of P such as for each $i = 1, 2,..., n,$ we have the following properties:

- if a predicate symbol is positive in S_i then its definition is contained in $\cup_{j \leq i} S_j$.
- if a predicate symbol is negative in S_i then its definition is in $\cup_{j < i} S_j$.

Each S_i is called a stratum. A stratification $P = S_1 \cup ... \cup S_n$ is maximal stratification if each stratum cannot be decomposed into different strata. A Stratified DataBase (SDB) is modelled by a stratified program [1].

3. The SDB Update Problem

With an aim of showing the difficulties of the update operations in SDB, we propose to study these operations and to study their influences on the computation of the model of the base and its stratifibility.

The rule update is not as simple as the one of a fact. The fact update generates in the worst of the cases addition and/or deletes other facts. In reality, the rule update can generate update of all facts deducted by this rule which, in their turns, will generate the addition and/or delete of several other facts. We retail this in this section.

3.1. SDB Update Properties

Seen that a SDB is modelled by a stratified program, we showed in [7, 8] the following properties which one can apply directly to the SDB:

- An explicit operation of deletion of a deduced fact doesn't have a sense, since it will be always deduced starting from the rules. If one wants to remove a fact deduced from the SDB, it is necessary to remove the facts or the rules which were used to deduce it.
- An explicit operation of insertion of a deduced fact does not modify the minimal model of the program since this fact already exists in the model.
- An operation of update in the SDB (known as explicit update) starts one or several updates of the facts deduced (known as induced updates) because of the presence of the rules. The updating intervening on the deduced facts and their instances are not known. Consequently, to carry out an operation of updating in the SDB amounts carry out this the induced operation and all updating which should be determined.

The difficulty of an operation of updating lies in the search for an effective method which makes it possible to determine the induced updating exactly as the example shows it.

Example: consider the *SDB* modelled as follows:

$p_1(a)$; $p_1(b)$; $p_3(c)$; $p_2(x) \leftarrow p_1(x)$;

$p_3(x) \leftarrow \neg\ p_1(x)$; $p_4(x) \leftarrow \neg\ p_2(x)$

The standard model of this program is $M_P=\{p_1(a), p_1(b), p_2(a), p_2(b), p_3(c), p_4(c)\}$.

Consider the deletion of the fact $p_1(a)$ and let us try to calculate the new models $M_{P'}$. The deletion of $p_1(a)$

involves the deletion of the deduced fact $p_2(a)$ and the addition of the deduced fact $p_3(a)$. The deletion of the deduced fact $p_2(a)$ involves the addition of the deduced fact $p_4(a)$. Hence, the new model is $M_{P'} = \{p_1(b), p_2(b), p_3(a), p_3(c), p_4(a), p_4(c)\}$.

3.2. Effects of UPD Operations on Stratifiability

Following an UPD operation, we should first of all be sure that the resulting database remains stratified. Then it is necessary to define the relation between the maximum stratification of the program *P* initial modelling the database and that of the object program P' modelling the resulting database, knowing the operation of UPD considered. Tables 1 and 2 show the influence of the UPD on the stratifiability in the case of facts and rules insert or delete.

Table 1. Insertion operation effects on stratifiability.

Update Operation	Consequences	Stratifiability
Fact q(a,b): the redicate q is not defined in the program P.	Stratum creation.	Yes
Fact q(a,b): the redicate q is already defined in the program P.	Stratum modification.	Yes
Clause: the predicate of the head appears for the first time.	Stratum creation.	Yes
Clause: the predicate of the head is already defined.	If the program is still stratifiable : - Stratum modification. - Or fusion of several strata.	Depends on cases

Table 2. Deletion operation effects on stratifiability.

Update Operation	Consequences	Stratifiability
Fact : the fact is omposing a stratum.	Stratum removal.	Yes
Fact : the fact belongs to a stratum composed of several clauses.	Stratum modification.	Yes
Clause: the clause composes the stratum.	Stratum removal.	Yes
Clause: the clause belongs to a stratum composed of several clauses.	- Stratum modification. - Or the stratum is splited into several strata.	Yes

According to the above study, we notice that an operation of UPD of a fact can involve the creation of a new stratum, the modification or the remove of a stratum. The rule update can generate the same consequences that the fact update, with in addition the fusion of several strata in only one stratum or the bursting of a stratum in several under-strata. It gives back more complex incremental computation of the M_P model.

To calculate the model $M_{P'}$, witch is the result of an update operation; several works have used sets of predicates, called supports [1, 3, 7, 8]. These supports are used to determine the part to remove from the model M_P after an update operation. The choice of the supports must minimize the number of migrations facts and the maintenance cost of these supports. The ideal is to have a support which determines exactly the facts which must be removed from the model to avoid the total migrations of facts [1, 3, 7, 8].

4. SEPN Formalism

In this section, we present an extension Predicates nets to support the DDB and SDB.

4.1. SEPN Nets

In this section, we describe the SEPN formalism. For further details concerning this approach, readers can refer to [3, 7, 8, 11]. An SEPN is defined by:

- A quintuple $N = (P, T, C, V, K)$, where P, T, C, V and K are respectively the set of places, the set of transitions, the set of colours, the set of variables and the set of constants.
- Two relations α and β, where β is a finite subset of $T \times P$ which elements are called unsigned arcs and α is a finite subset of $\{+,-\} \times P \times T$ which elements are called signed arcs (α+ positive arcs set and α- negative arcs set).
- Two applications $I\alpha$ et $I\beta$ defined by:
  $I\alpha : \alpha \rightarrow Z[V \cup K]$
  $I\beta : \beta \rightarrow Z[V \cup K]$

where $Z[V \cup K]$ is the set of finite formal combinations of $V \cup K$ elements. A set Garde, where Garde(t), t being a transition, imposes firing conditions between tokens contained in input places. A bijective application Cl from T to C, which associates a colour to each transition.

4.2. Dynamic Aspect of the SEPN

In a SEPN net, tokens are colored [8, 11]. A colored token is an element of the set $K^n \times P(C)$, where $P(C)$ is a set of parts of C. A colored token j has then this form:

$$j = ((x_1, x_2, \ldots, x_n), Col) \quad (1)$$

where $(x_1, \ldots, x_n)$ is the argument of j $(arg(j))$ and Col its path $(path(j))$. The path is a set of colors saving the history deduction.

We define the following operations on the elements of the set $K^n \times P(C)$:

- Equality: $j = j' \Leftrightarrow arg(j) = arg(j')$ and $path(j) = path(j')$
- Order relation $\leq$: $j \leq j' \Leftrightarrow arg(j) = arg(j')$ and $path(j) \subseteq path(j')$
- Subtraction: if $arg(j) = arg(j')$ then $j - j' = (arg(j), path(j) \setminus path(j'))$.

A token of the form $j=(arg(j),\{\varnothing\})$ is called neutral token. Consequently, we have these two results (1) the subtraction to a token from itself gives a neutral token and (2) the addition operation is idempotent ($j + j = j$).

Let R be an SEPN net and Mo a function from P to $K^n \times P(C)$. The function Mo is called initial marking of R. This function associates, initially, to each place of R a finite set of colored tokens. The SEPN marking M is defined by the association to each place a finite set of tokens.

The transition firing process in SEPN is different from ordinary Petri nets. In fact, the production of new tokens happens without removing the tokens used while firing the transition. We make a distinction between valid transitions from fireable transitions.

If the new token already exists in the destination place, it will not be regenerated. This transition is then fireable but not valid. If the token does not exist in the output place, the transition is valid and is fired. By this way, an SEPN can not contain double tokens. Each place p is $k^{m(p)}$ bounded, where $m(p)$ is the marking of p and k is the cardinality of the set K.

5. Correspondence Between SDB and SEPN Nets

The SEPN formalism allows us to build efficient algorithms for checking stratifiability, determination of the maximal stratification, the computation of the standard model, and management of update operations on facts and clauses (explicit and induced updates). Indeed, we established a correspondence between the SEPN formalism and SDB. This correspondence is presented in Table 3. Due to space limitations, we do not demonstrate this correspondence.

Table 3. Correspondence between SDB and SEPN.

SDB	SEPN
A predicate p	A place p
A fact q(a,b)	A token in the place q
A clause r	A transition t
Link between the predicates'variables and the clause body	Garde of the transition
Standard model of the program	Net marking
Program stratification	Net stratification
Program strata	Net strata
Addition or removal of a fact	Addition or removal of a token
Addition or removal of a clause	Addition or removal of a transition and its related arcs
An integrity constraint	A transition tc

5.1. Stratifiability Study and Maximal Stratification Determination Using SEPN

Stratifiability checking is released after the definition of the logical program and after each update operation. This consists on detecting the presence of a negative circuit in the SEPN. Once the stratifiability of a program is checked, it is necessary to find out the maximal stratification in order to compute the standard model. For this purpose, we give the following definitions:

Definition 1: let R be a SEPN and $V = \{p1, T1..., pn, tn\}$ (such as $\{p1..., pn\} \in P$ and $\{T1.., tn\} \in T$) a strongly connected component of R. We define a stratum of R as a strongly connected component V of R with one of following conditions:

- Card(V) >1
- Card(V)=1 and V = {p}, p ∈ P, with M(p) ≥ 0 or ∃ Ti∈T / (Ti, p)∈β.

where M(p) is the marking of the place p. After determining the SEPN strata, we put an order on them. We introduced, for this purpose, the concept of reduced graph.

Definition 2: let G be the set of the SEPN strata. The reduced graph of the SEPN is the graph $Gr = (X, U)$, where:
Each node of X represents a stratum.
And $U = \{(S_i, S_j), i \neq j \mid \exists p \in S_i$ and $T \in S_j \mid (p,t) \in P\}$, where U is a finite set of arcs connecting strata.

An arc of the reduced graph is positive (respectively negative) if there is a positive arc (respectively negative) in the SEPN relating a place of S_i and a transition from S_j. The reduced graph of an SEPN does not contain circuits. It represents dependences between strata.

Definition 3: let R be a stratified SEPN and $V_1, ..., V_n$ a maximal stratification of R. In the stratified firing process, the firing of a transition in V_i, $i \in [1..n]$, starting from a given marking of SEPN, can be made only if the firing of all the transitions belonging to V_j, $j \leq i-1$, is done.

Example: let us consider the following DDB modelled as follows:
F(f, c) ← ; G(n, p) ← ;
A(x, z) ← C(y, z); C(x, z) ← A(y, z), F(x, y);
D(z, y) ← G(z, y), not (C(x, z)) ;
G(x, z) ← D(x, y), G(y, z);

Figure 1 shows the SEPN corresponding to DDB. The DDB is stratifiable since its SEPN does not contain recursion through negation.

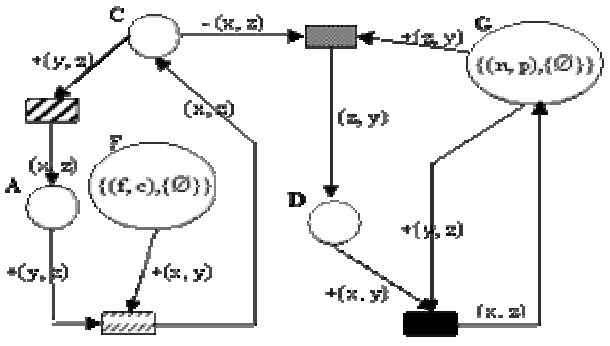

Figure 1. SEPN net of the P.

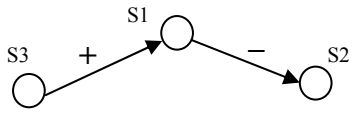

Figure 2. Gr Reduced graph of DDB.

The reduced graph of *P* is shown in Figure 2. It shows the maximal stratification of P, which is composed of these strata:

S1 { A(x, z) ← C(y, z);
C(x, z) ← A(y, z), F(x, y);

S2 { G(n, p) ← ;
D(z, y) ← G(z, y), not(C(x, z)) ;
G(x, z) ← D(x, y), G(y, z);

S3 { F(f, c) ← ;

5.2. Update Operation Optimization in SEPN

5.2.1. Update Operation Optimization of Facts

The addition of a token in a place *p* may lead to: (1) the addition of other tokens in the places related positively to *p* and (2) the removal of tokens from the places negatively related to *p*. In opposition, the removal of a token of a place *p′* may lead to: (1) the addition of tokens in the places negatively related to *p′* and (2) to the removal of tokens from the places positively related to *p′*.

The use of colored tokens has the advantage of saving the deduction history. This allows us to recognize the transitions used in the firing process. Thus, it is easy to reduce facts migration. In fact, the tokens that do not contain the transition's color related to the place, in their paths, will not be touched.

5.2.2. Update Operation Optimization of Clauses

The update of a clause is equivalent to the update of a transition in the PRES net. Knowing the corresponding sub-net of the clause and the reduced graph, we find out, directly, if the program remains stratifiable and its new maximal stratification. After this step, the problem is reduced to facts update, which is already solved.

6. VI_DBS Tools

VI_DBS implementation was carried out by using the concept of SEPN. Since that a SDB is modelled by a stratified program, we enriched the STRPRO tool [12], to support the concept of DDB, SDB, IC, and UPD and query evaluation. The architecture of the VI_DBS is made up mainly of the three following modules:

- The analysis module allows the lexical and syntactic analysis of the SDB or DDB. It permits the representation of the SDB by a stratified SEPN, the test of the stratifiability and the computation of the inherent stratification to the UPD. This module permits also the optimization of the extended predicate networks.
- The stabilization module allows applying the pair (SDB, UPD) in a network to extended predicate stratified with an initial marking, to calculate the steady marking of the network (model of the SDB). In addition, it updates the Stratified SEPN and its marking.
- The transaction module is in charge of the following actions: to filter and to increase the network to predicates marked in the goal to define a view bound to the UPD and the database (via his network of representation) and to value some queries.

6.1. VI_DBS Implementation

In order to help understanding the stratification concept, we built VI_SDB tool. This tool is plate-form independent since it is developed with Java. First, we selected an open source tool Petri tool [4], used in editing and simulating ordinary Petri nets. Then, we made up required modifications to adapt it to SEPN concept. After that, we added necessary modules for the manipulation of stratified database. Finally, we added layers of graphical user interfaces.

VI_SDB is composed of two interfaces as shown in Figures 3 and 4. The first one allows the edition of the DDB in textual mode as shown in Figure 3 and the second one allows the edition of the DDB in graphical mode as shown in Figure 4. The textual interface is composed of a menu bar and four panels. The program edition is done in the edition panel. The *Gr* panel holds the reduced graph. The strat panel contains the composition of each stratum. The Status Panel displays messages to users (stratification result, compilation results…).

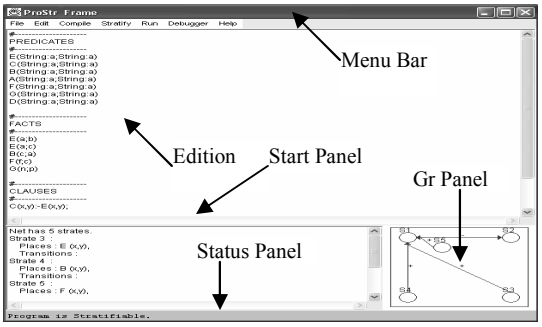

Figure 3. VI_SDB Interfaces textual user interface.

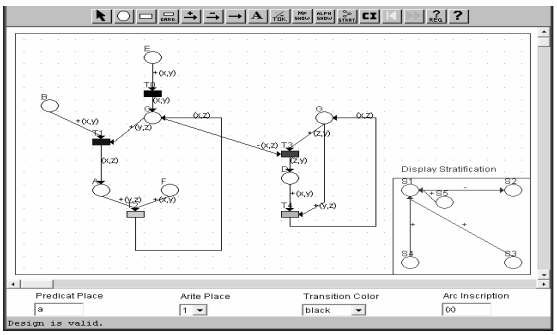

Figure 4. VI_SDB Interfaces graphical user interface.

VI_SDB is composed of nine main modules:

- Editor module: VI_SDB offers two different ways for the edition of a program describe a database: textual and graphical. In addition, users can save and reload their programs into and from files.
- Compiler module: the edition of database in textual mode should be followed by compilation. This operation consists on the syntax analysis of the given database and its mapping - in case of validity - into its corresponding SEPN.
- Stratifiability checker module: VI_SDB contains a module that checks the stratifiability property of a given database. This operation consists on the verification of absence of recursion through negation in the SEPN.
- Maximal stratification extractor module: once the database is stratifiable, user can call the "Stratify" command in order to extract the reduced graph Gr.
- Standard model computation module: After the stratification of the database, we can compute its standard model.
- Query evaluation module: since the standard model corresponds to net marking, query evaluation in VI_SDB is simply the token existence test.
- Update operations module: after each update operation, the tool checks the database stratifiability. In case of validity, the new maximal stratification is determined. After that, the standard model is updated.
- Integrity constrains manager: this module allows the edition of integrity constrains. While the DDB contains integrity constrains, after each fact update operation, the corresponding constrains are checked. In case of non validity any IC, the initial update operation is cancelled.
- The debugger module: VI_SDB provides users with the ability to debug their programs describe database by the means of the debugger. The debugger is the graphical interface that presents the internal modelling of a given program (its corresponding extended predicate net). Using this interface, we can perform all already cited operations.

Example: let a DDB describes family's tree:

Father (Ali, Saleh) ← ;
Father (Mohamed, Ali) ← ;
Mother (Alia, Saleh) ← ;
Female (Alia) ← ;
Age (Ali, 35) ;
Parent (x, y) ← Father (x, y);
Parent (x, y) ← Mother (x, y);
Ancestor (x, y) ← Parent(x, y);
Ancestor (x, y) ←Parent(x, y), Ancestor (x, y);

We suppose that we have the following integrity constraints:

- An individual cannot have two fathers,
- An individual cannot have two mothers,
- The Age of any person is small than 150

These constraints are modelled as follows:

- x = z ← Father(x,y) , Father(z, y)
- x = z ← Mother(x, y) , Mother(z, y)
- (y<150) ← Age(x,y)

The corresponding model with VI_SDB is shown in Figures 4 and 5.

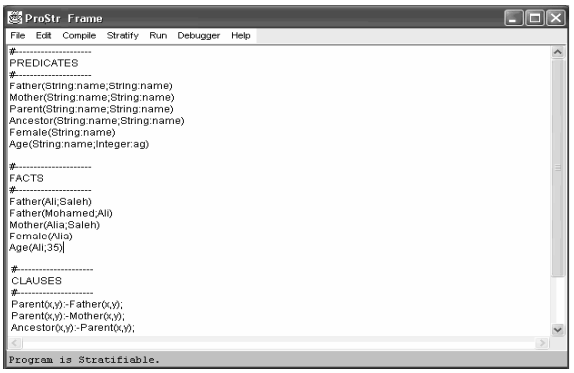

Figure 5. Family's tree example implemented with VI_SDB implementation with the textual interface.

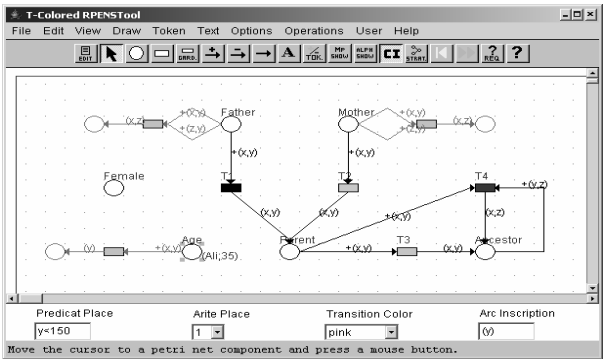

Figure 6. Family's tree example implemented with VI_SDB implementation with the graphical interface.

7. Conclusion

Even though DDB are based on a well established formalism, which is the first order logic, DDB didn't meet the expected success. The DDBMS was mainly used for academic purposes. Indeed, the DDBMS are judged abstract, absent from commercial offers, and expensive. This is due to many reasons that are relative to:

- The definition of a deductive and/or stratified database.
- The study of the stratifiability.
- The determination of the maximal stratification.
- The incremental definition of strata.
- The checking of integrity constraints.

These problems are more important when the database is voluminous. In this paper, we proposed a new and

convivial approach that aims to make the DDB manipulation non abstract and easy. This approach permits to simplify the concepts already mentioned. The proposed system supports rules update and is not limited to facts updating as in known deductive systems. We think that our tool VI_SDB is suitable as a learning tool of DDB and SDB abstract and delicate concepts.

As future work, we plan to extend: first the extended SEPN with object oriented concepts, then our tool VI_SDB to handle object oriented DDB and SDB.

References

[1] Apt R. and Pugin J., "Management of Stratified Databases," *Technical Report CS-TR-87-41*, University of Texas at Austin, USA, 1987.

[2] Apt R., *Handbook of Theoretical Computer Science*, Elsevier, 1991.

[3] Barkaoui K., Boudrigua N., and Touzi A., "A Transition Net Formalism for Deductive Databases Efficiently Handling Quering and Integrity Constrains," *in Proceedings of Database and Expert Systems Applications (DEXA)*, Spain, p. 221-225, 1992.

[4] Brink S., A Petri Net Design Simulation and Verification Tool, http://www.csh.rit.edu/~rick, 1996.

[5] Bry F., "Logic Programming as Constructivism: A Formalization and its Application to Databases," *in Proceedings of Principles of Database Systems (PODS)*, USA, pp. 34-50, 1989.

[6] Gardarin G., Bases de Données Objet et Relationnel, *Eyrolles*, 2000.

[7] Grissa A., "Contribution à l'Etude, à la Conception et au Prototypage des Bases de Données Déductives," *PhD Thesis*, Tunisia, 1994.

[8] Grissa A., Jerad C., and Ounelli H., "New Approach for Manipulation of Stratified Programs," *World Enformatika Conference*, USA, pp. 256-259, 2005.

[9] Grissa A. and Ounelli H., "XDatalog: A Language for Deductive and Stratified Databases," *International Journal of Computer Science and Network Security*, vol. 5, no. 10, pp. 84-93, 2005.

[10] Jager G. and Stark R., "The Defining Power of Stratified and Hierarchical Logic Programs," *Computer Journal of Logic Programming*, vol. 1, no. 3, pp. 55-77, 1993.

[11] Jerad C., "Outil d'Analyse des Bases de Données Déductives Formulées à l'Aide des Réseaux à Prédicats Etendus Stratifiés," *Master Memory*, Tunisia, 2003.

[12] Jerad C., Grissa A., and Ounelli H., "STRPRO Tool for Manipulating Stratified Programs Based on SEPN," *in Proceedings of World Enformatika Conference*, Turkey, pp. 252-255, 2005.

[13] Kiernan G. and De Maindreville C., "The RDL/C Language Reference Manuel V1," *Technical Report N°123*, 1990.

[14] Kolaitis G. and Vardi Y., "On the Expressive Power of Datalog: Tools and a Case Study," *Journal of Computer and System Sciences*, vol. 51, no. 1, pp. 110-134, 1995.

[15] Lloyd W., *Foundations of Logic Programming*, Springer-Verlag, 1987.

[16] Mengchi L., "Deductive Database Languages: Problems and Solutions," *Computer Journal of Computing Surveys*, vol. 31, no. 1, pp. 27-62, 1999.

[17] Minker J. and Apt R., *Foundations of Deductive Databases and Logic Programming*, Morgan Kaufmann Publishers, 1988.

[18] Ramakrishnan R., Srivastava D., Sudarshan S., and Seshadri P., "Implementation of the CORAL Deductive Database System," *in Proceedings of ACM SIGMOD on the Management of Data*, United States, pp. 167-176, 1993.

[19] Ramamohanarao K. and Harland J., "An Introduction to Deductive Database Languages and Systems," *The International Journal on Very Large Data Bases* (*VLDB)*, vol. 3, no. 2, pp. 107-122, 1994.

[20] Touzi A. and Barkaoui K., "Un Formalisme de Modélisation et d'Optimisation des Bases de Données Déductives Basé sur la Théorie des Réseaux de Petri de Haut Niveau," *2nd Maghrebine Conference on Software Engeneering and Artificial Intelligence*, Tunis, pp. 99-115, 1992.

[21] Ullman D., *Principles of Database and Knowledge Base Systems*, Computer Science Press, 1988.

[22] Ullman D., "Implementation of Logical Query Languages for Databases," *Computer Journal of Transactions on Database Systems*(*TODS*), vol. 10, no. 3, pp. 289-321, 1986.

Amel Touzi received the Diploma of engineering in computer science and PhD in computer science from the Faculty of Sciences of Tunis, Tunisia in 1989 and 1994, respectively. Currently, she is an assistant professor at the Department of Technologies of Information and Communications in the National School of Engineering of Tunis.