The Constraint Operator of MedLan: its efficient implementation and use

Patrizia Asirelli (Contact author)
Istituto di Elaborazione della Informazione - CNR
Via S. Maria, 46, I-56126 Pisa, Italy, tel. +39-50-593477, fax. +39-50-554342, email: asirelli@iei.pi.cnr.it

Chiara Renso and Franco Turini
Dipartimento di Informatica, Universita di pisa, Corso Italia, 40, I-56125 Pisa, Italy, tel. +39-50-88743, fax. +39-50-887226, email:turini@di.unipi.it, renso@sias.it

Abstract
A declarative mediator language, based upon operations among logic theories is introduced. In particular we concentrate on the *constraint* operator. The denotational semantics of the language is introduced together with the definition of a bottom-up efficient implementation. The use of the constraint operator for security within a mediator architecture for database integration is suggested and presented by means of a simple example.

Keywords
database integration, federation, mediators, constraints, multilevel security, internal security

1 INTRODUCTION

The Internet and the World Wide Web capability are showing the need for organizations to access and integrate different sources of information. Future applications are likely to be built by putting together systems developed and managed at different sites. Integration, federation, cooperation etc of information sources or software systems, in general, seem to be a must. Security and privacy are thus becoming more crucial (Jajodia 1996a), (Jajodia 1996b), both for relational and advanced database systems.

*work partially funded by Progetti Coordinati CNR-Comitato 12 : "KINE-Knowledge Integration Environment" and "Programmazione Logica (Logic Programming)" and EADTN Project: ERCIM Advanced Database Technology Network, EEC contract n. CHRX-CT94-0531.

Given the state of the art, providing for semantic heterogeneity and for "secure" integration of information sources/databases is crucial for the development of future distributed/integrated applications.

Semantic heterogeneity and integration of databases is a hot area of research. Various architectures can be found in the literature, as it is in (Hull 1997), where a *mediator* (notion due to Wiederhold (Wiederhold 1992)) turns out to be a very interesting and promising one.

Mediation supports semantic integration of databases providing for read-only view of information sources that reside on different sites and, in some cases, update capabilities.

Interesting proposals for mediating database systems can be found in the literature, both for relational and logic/deductive databases (Papakostantinou, Garcia-Molina and Ullman 1996, Subrahmanian 1994, Lu *et al* 1995, Aquilino *et al.* 1995, Asirelli, Renso and Turini 1996).

With respect to security of systems, many models, policies and enforcing mechanisms can be found in the literature, as it is summarized in (Bertino, Samarati and Jajodia 1997) and (Jajodia *et al.* 1997). Security, within the framework of deductive databases and their integration has also received great attention (Bertino, Jajodia and Samarati 1995, Bonatti, Kraus and Subrahmanian 1995, Candan, Jajodia and Subrahmanian 1996). In relation to a mediator approach to security we also mention the TIHI project (Wiederhold *et al* 1996a, Wiederhold *et al* 1996b).

In this paper we consider an approach to build a federation of information sources via mediators. The language we refer to is MedLan (Aquilino *et al.* 1995, Asirelli, Renso and Turini 1996, Aquilino *et al.* 1997) that is an extended logic language for deductive databases where the basic extensions are the partitioning of the deductive database into a collection of theories, operators to combine them, and the "in" feature, that is a sort of "message passing" feature. MedLAn has been given an operational and a denotational semantics.

In this paper we particularly concentrate on the definition of a kind of seminaive implementation of the MedLan Language, where the most interesting aspect is the constraint (/) operator. This operator allows for different applications (Renso 1998, Aquilino *et al.* 1997, Asirelli*et al.* 1998) that we have studied and developed. Here we present, as an example of the use of the constraint operator of MedLan, an approach to database security within a mediator architecture for database integration. For the moment we have taken into consideration a multilevel security model, as the Bell-La Padula model, where the data and the users are classified into various classes (or levels) and then the appropriate security policy of the organization is implemented.

The general idea is to use the language MedLan to build a set of logical theories that constitute a middle layer of an architecture to build an application which uses a federation of databases as the source of information. In other words, MedLan can be used to implement the layer of an application, in an integrated environment, that stands between the database sources and the final user that will be allowed to see and use only part of the complete set of information. In this way we are able to address two unusual aspects, at the same time: the semantic integration between database sources and the implementation of security policies, where the "constraint" operator plays the major role.

Section 2 introduces the syntax of the language, an intuitive explanation of composition operators, and an abstract semantics. In section 3 we discuss an implementation of MedLan based on an extension of the semi-naive computation rule for deductive data bases. The complete proof is not included here for shortage of space, but can be found in (Renso 1998). In section 4, an example of its application to security is given. Finally, in section ?? we conclude.

2 THE MEDLAN LANGUAGE

We consider a set of meta-level operations for composing definite logic programs, originally introduced in (Brogi 1993, Brogi*et al.* 1994, Aquilino *et al.* 1995) *Union* ($\cup$), *Intersection* ($\cap$), and *Constraint* ($/$).

MedLan is the *language of program expressions* defined by these operations as follows:

$$Pexp ::= Program \mid Pexp \cup Pexp \mid Pexp \cap Pexp \mid Pexp/Program$$

where *Program* is a named collection of clauses. Each set of clauses (program) is associated with a unique name by means of a global naming mechanism.

In the sequel we will abuse the notation and use a program identifier to directly denote the set of clauses associated with it.

More precisely, a program is a finite set of extended definite clauses of the form

$$A \leftarrow B_1, \ldots, B_n$$

where each B_i is either an atomic formula or a meta-level formula of the form "*C in Pexp*", where C is an atomic formula and *Pexp* a program expression. A goal like "*C in Pexp*" introduces a form of message passing between object level program. The idea is that the program containing the goal "*C in Pexp*", sends the message C to the "virtual" program denoted by "Pexp". As usual, logical variables act as input/output channels between programs.

We assume that the language in which programs are written is fixed. Namely, there is a fixed set of function and predicate symbols that include all function and predicate symbols used in the programs being considered. Moreover, program names and program composition operations are disjoint from all other constant and function symbols that may occur in programs.

Here we give the operators an informal semantics by means of examples. In section 2.1 we will show the formal (abstract) semantics.

Consider the following programs P and *Res_Dept*:

P:
  $can\_access\_folder(a\_inform, X) \leftarrow employee(X)$ *in* Res_Dept
  $employee(john) \leftarrow$
  $employee(ann) \leftarrow$

Res_Dept:
  $employee(mary) \leftarrow$

The first rule of P states that "a person can access a particular folder a_inform if he/she works in the research department, *Res_Dept*"

The query $can\_access\_folder(a\_inform, X)$ in the program P bounds the variable X to *mary* since the evaluation of the goal $employee(X)$ is performed in the program *Res_Dept*.

Given a program expression E, we show a plain logic program that behaves as the program expression, i.e. it provides the same answers to the same queries, whatever is the operational semantics in use. We refer sometimes to this program as to the *virtual* program denoted by the expression. Such a transformational approach, is useful for an intuitive understanding of program expressions.

Consider the following plain programs:

P:
  $can\_access\_folder(a\_inform, X) \leftarrow$
    $employee(X)$ *in* $(Res\_Dept \cup Direction\_Dept)$,
    $has\_authorization(X)$ *in* Aut_Module
  $employee(john) \leftarrow$
  $employee(ann) \leftarrow$

Res_Dept:
  $employee(mary) \leftarrow$
  $employee(fred) \leftarrow$

$Direction\_Dept$:
  $employee(john) \leftarrow$
  $employee(ann) \leftarrow$

Aut_Module:
  $has\_authorization(mary) \leftarrow$
  $has\_authorization(john) \leftarrow$

Here, P gives access to folder *a_inform* to people working either in the Res_Dept or in the Direction_Dept. That is, "mary", "fred", "john" and "ann". Because of the further condition that the person must also be authorized, as specified in the authorization module Aut_Module, the answer to the query *can_access_folder*$($*a_inform*$, X)$ in the program P for the variable X will be: *mary* and *john*.

The $\cup$ operator makes the program denoted by the program expression *Res_Dept* $\cup$ *Direction_Dept* behaves as a plain program containing the clauses of *Res_Dept* **and** the clauses of *Direction_Dept*. As the example shows, union may be used to factor knowledge in different modules.

Intersection allows to combine knowledge by merging clauses with unifiable heads into clauses having the conjunctions of the bodies of the original clauses as body. The net effect is that the two plain programs act as sets of constraints one upon the other.

Consider the following example:

```
P:
  can_access_folder(a_inform, X) ←
        employee(X) in (Res_Dept ∪ Direction_Dept),
        has_authorization(X) in (Aut_Module ∩ Validity_of_Aut)
  employee(john) ←
  employee(ann) ←

Res_Dept:
  employee(mary) ←
  employee(fred) ←

Direction_Dept:
  employee(john) ←
  employee(ann) ←

Aut_Module:
  has_authorization(mary) ←
  has_authorization(john) ←

Validity_of_Aut:
  has_authorization(mary) ←
  has_access(john, folder_B, section_E) ←
  has_access(X, Y, Z) ← director_of(Z, X)
```

Now the answer to the query *can_access_folder*$($*a_inform*$, X)$ in the program P will give for the variable X the binding *mary* whose authorization is still "valid".

Notice that *Aut_Module* $\cap$ *Validity_of_Aut* does not say anything about *has_access*, since nothing about *has_access* is deducible from *Aut_Module*.

The constraint operator combines the features of union, intersection and a simple form of negation to provide an *asymmetric* composition between a program P and a program Q where Q acts as a set of constraints for P as it is illustrated by the following example. Consider

$Constraint\_module$
$\quad has\_authorization(X) \leftarrow was\_authorized\_by(Y, X), is\_director(Y, Z),$
$\qquad employee(X, Z)$ in $Wrapper\_Dept$

$Wrapper\_Dept$:
$\quad employee(X, res\_dept) \leftarrow employee(X)$ in Res_Dept
$\quad employee(X, direct\_dept) \leftarrow employee(X)$ in $Direction\_Dept$

$Validity\_of\_Aut$:
$\quad has\_authorization(mary) \leftarrow$
$\quad has\_access(john, folder\_B, section\_E) \leftarrow$
$\quad has\_access(X, Y, Z) \leftarrow director\_of(Z, X)$

The following plain program behaves as the program expression $Constraint\_module/Validity\_of\_Aut$.

$\quad has\_authorization(X) \leftarrow X \neq mary, was\_authorized\_by(Y, X),$
$\qquad is\_director(Y, Z), employee(X, Z)$ in $Wrapper\_Dept$
$\quad has\_authorization(mary) \leftarrow was\_authorized\_by(Y, mary), is\_director(Y, Z),$
$\qquad employee(mary, Z)$ in $Wrapper\_Dept$

While the following plain program behaves as the program expression: $Validity\_of\_Aut/Constraint\_module$.

$\quad has\_authorization(mary) \leftarrow was\_authorized\_by(Y, mary), is\_director(Y, Z),$
$\qquad employee(mary, Z)$ in $Wrapper\_Dept$
$\quad has\_access(john, folder\_B, section\_E) \leftarrow$
$\quad has\_access(X, Y, Z) \leftarrow director\_of(Z, X)$

Notice that the constraint is applied only to mary while the remaining knowledge in the module is not affected.

2.1 Denotational semantics

Now we give an abstract semantics. In (Renso 1998, Aquilino *et al.* 1995, Aquilino *et al.* 1997) also an operational top-down semantics is given and it is shown that the two semantics coincide. The semantics is limited to positive deductive data bases, and it is given in a bottom-up style by extending the standard immediate consequence operator ($T(P)$).

Recall that, for a definite logic program P, the immediate consequence operator $T(P)$ is a continuous mapping over Herbrand interpretations defined as follows (van Emden and Kowalski 1976). For any Herbrand interpretation I:

$$A \in T(P)(I) \iff (\exists \mathbf{B} : A \leftarrow \mathbf{B} \in ground(P) \wedge \mathbf{B} \subseteq I)$$

where $\mathbf{B}$ is a (possibly empty) conjunction of atoms and $ground(P)$ denotes the ground (i.e. fully instantiated) version of program P.

Such an approach is motivated by the observation that the classical least model semantics is not compositional. That is, it is not possible to obtain the least model of, say, the union of two programs P and Q by homomorphically composing the least models of P and Q. In (Brogi 1993) it is shown that the T_P-based semantics is in fact both compositional and fully-abstract with respect to the repertoire of composition operations adopted in this paper.

Definition 1 *The semantics of program expressions is given as follows:*

$$T_{(E \cup F)}(I) = T_E(I) \cup T_F(I)$$

$$T_{(E \cap F)}(I) = T_E(I) \cap T_F(I)$$

$$T(E/F)(I) = T(E \cap F)(I) \cup (T(E \backslash\backslash F)(I)$$
$$T(E \backslash\backslash F)(I) = T(E) \backslash \{A \mid A \leftarrow G \in Ground(Q)(\mathcal{B})\}$$

The immediate consequences of a program expression E constrained by a program F is a combination of the union and intersection operator and a kind of *complement* of a program w.r.t. a program expression. Informally, consider the case in which E is a plain program constrained by a set of clauses F. The resulting program is obtained by the union of two parts. One is the intersection of the two programs, that forces them to agree during the deduction. But intersection alone is not enough, because some clauses would be missing in the result. In particular, we miss all the clauses for predicates which are defined in E and not constrained by F. These predicates are of two kinds: the ones which do not have a definition in F, and those which have a definition in F that constrains only a subset of atoms potentially derivable in E.

3 IMPLEMENTATION ISSUES

Here we show how the T(P) semantics for our operators can be turned into an efficient bottom-up strategy by exploiting the technique of the so-called semi-naive computation strategy, that allows one to avoid redundant computations of recursive rules.

The standard implementation of deductive databases is based on the bottom-up evaluation of logic programming, i.e. on efficient implementations of the computation of the fixpoint of the immediate consequence operator. In this section we show how the simplest of these efficient implementations, i.e. the semi-naive computation strategy, can be extended to handle the new features of MedLan. We consider here only positive programs. Not even the extension to stratified databases is straightforward, given that, for example, the union of two stratified databases is not necessarily stratified. The definition of classes of stratified programs that can be handled by a semi-naive computation strategy is the subject of our current research.

3.1 The Seminaive Evaluation Technique

The seminaive evaluation technique (Abitebul, Hull and Vianu 1995, Ullman 1988) is a straightforward extension of the immediate consequences operator (also called *naive* evaluation), in that it avoids the recomputation of the same atoms, that might be triggered by recursive definitions, by focusing the computation only on the new atoms generated in the last step.

We define the seminaive computation as an extension of the standard T(P) operator. Notice that we refer to the T(P) semantics described in section 2.1 where we do not take into account the *in* feature. Extending the seminaive definition to include the message passing mechanism mirrors the approach presented in (Brogi, Renso and Turini 1997) and is matter of future studies.

The new operator $\tau(P)$ has two arguments. The first one represents the current interpretation, the second one, Δ, represent the set of facts computed in the previous step.

Definition 2 *Let P be a program, I, Δ interpretations, then $\tau : (2^B \times 2^B) \to (2^B \times 2^B)$ is defined as follows*

$$\begin{aligned}\tau(P)(I,\Delta) = (\{I \cup \Delta\},\ \{A \mid\ & \exists B_1,\ldots,B_n, A \leftarrow B_1,\ldots,B_n \in ground(P) \\ & \wedge \exists i \in [1,\ldots,n] : B_i \in \Delta \\ & \wedge \{B_1,\ldots,B_n\} \subseteq I \cup \Delta\} - (I \cup \Delta))\end{aligned}$$

The powers of τ are defined as usual

Definition 3 *Powers of τ*

$$\begin{aligned}\tau(P)(\emptyset,\emptyset) \uparrow 0 &= (\emptyset,\emptyset) \\ \tau(P)(\emptyset,\emptyset) \uparrow i &= \tau(P)(\tau(P)(\emptyset,\emptyset) \uparrow (i-1))\end{aligned}$$

3.2 The Seminaive for Composition Operators

The extension of the τ operator for the composition operators is quite intuitive. The bottom-up step of the seminaive function applied to $P \cup Q$ is the set-theoretic union of the respective arguments, and an analogous definition works for intersection.

$$\tau(P \cup Q) \quad = \tau(P)(I,\Delta) \sqcup \tau(Q)(I,\Delta)$$

$$\tau(P \cap Q) \quad = \tau(P)(I,\Delta) \sqcap \tau(Q)(I,\Delta)$$

where
$(A,B) \sqcup (A',B') = (A \cup A', B \cup B')$
$(A,B) \sqcap (A',B') = (A \cap A', B \cap B')$

In the definition of the constraint operator we will use the *Ground* predicate with two arguments: the first one is a given program, and the second is the Herbrand base with respect to which we instantiate the given program.

$$\tau(P/Q)(I,\Delta) \quad = \quad \tau(P \cap Q)(I,\Delta) \sqcup \tau(P \setminus\setminus Q)(I,\Delta)$$

The definition of τ for the constraint operator has the same structure of the naive definition based on the immediate consequence operator. Since τ works on pairs of interpretations we need to use the operator $\sqcup$, that performs the union of pairs.

$$\begin{aligned}&\tau(P \setminus\setminus Q)(I,\Delta) \; = \\ &(\tau(P)(I,\Delta)_1 \setminus \{A \mid A \leftarrow B_1,\ldots,B_n \in \mathit{Ground}(Q)(\mathcal{B}_{\mathcal{P}}), \\ &\tau(P)(I,\Delta)_2 \setminus \{A \mid A \leftarrow B_1,\ldots,B_n \in \mathit{Ground}(Q)(\mathcal{B}_{\mathcal{P}})\}\end{aligned}$$

The auxiliary definition of $\tau(P \setminus\setminus Q)$ is such that, given a pair (I,Δ), it computes a new pair, containing atoms that can only be computed by using P. In (Renso 1998) the correctness of the semi-naive definition has been proved, as stated by the following theorem.

Theorem 1 *The seminaive definition for the composition operators is correct w.r.t the T(P) semantics of the operators. Let E be a program expression:*

$$(\tau(E) \uparrow \omega)_1 = T(E) \uparrow \omega$$

4 AN EXAMPLE FOR INTERNAL SECURITY

The general idea of an architecture of a system for integrating database sources, according to a given set of security rules, that we have in mind is depicted in Fig.1.

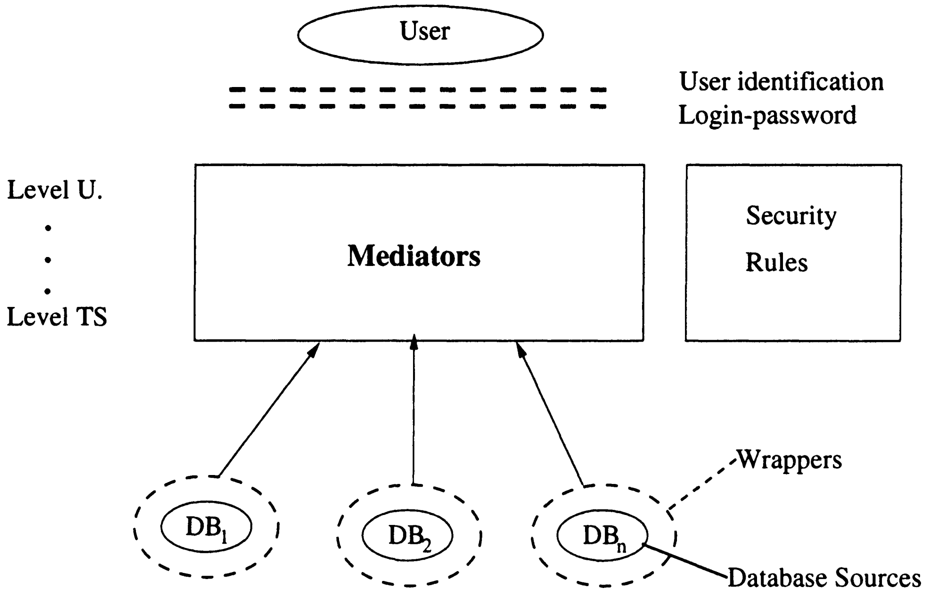

Fig.1 - A general architecture

The architecture can be considered as consisting of three levels:

- the first level hides those parts of the database source that we do not want to be visible outside; wrappers are defined for each database and they export only those parts of the databases that we allow to be visible to someone; each integrated database is assumed to be provided with secure transmission system and firewall. We only expect to interface with the underline database systems,(e.g. it can be any Java interface) that only provide us with a number of relations that can be queried according to the internal policies of the integrated database.
- the second security level concerns the mediators layer. Here mediator modules use data from wrappers to reason on it and to perform semantic integration; Mediators can use data provided also by other wrappers and mediators. At this level, more policies can be implemented. They can depend on the application being implemented or it can be stated by agreements with the particular integrated database. That is, some exception to the internal

policy of a database can be allowed from the integrated database as long as some rule in the integrated environment are satisfied. As an example we can imagine that some pieces of information are made available to the integrated environment manager and to the other database managers (in the integrated environment) but they must be hidden to some external users etc.

- the third level of security is realized by a user identification system. It could be a login-password system or some other more sophisticated system, like a voice recognizer. No particular identification system is assumed in our models.

Thus our model concentrates on the integration layer to provide for security of the application and of the integrated systems, i.e. for the "internal" core of the integrated system.

The general architecture in Fig. 1 could be applied to the following situation of integrating different databases of employees with permission assigned for each level, as it is handled by mandatory access control models.

DB1
```
employee(john,rossi,001,dept1,100)
employee(susan,white,302,dept2,108)
...
```
DB2
```
employee(frank,green,527,70)
employee(mary,brown,670,65)
...
```
W_DB1
```
employee(CodRef,Dept,Salary) ←
   employee(FName,LName,CodRef,Dept,Salary)in DB1
...
```
W_DB2
```
employee(CodRef,nil,Salary) ←
   employee(FName,LName,CodRef,Salary)in DB2
...
```
Mediator1
```
employee(User,CodRef,Dept,Salary) ←
   user(User),employee(CodRef,Dept,Salary)in(W_DB1 ∪ W_DB2)
...
```
Mediator2
```
employee(User,CodRef,Dept,Salary) ←
   employee(User,CodRef,Dept,Salary)in(Mediator1/Security_Rules)
...
```
Security_Rules
```
employee(User,CodRef,Dept,Salary) ← has_permission(User)in Level_C
```

```
...
Level_U
has_permission(user1)
has_permission(user2)
...
Level_C
has_permission(user3)
has_permission(user4)
...
Level_T
has_permission(user5)
has_permission(user6)
...
Level_TS
has_permission(user7)
has_permission(user8)
...
```

Please note that DB1 and DB2 report on a set of employees that are represented, in the two databases, with a relation that has the same name but different number of parameters. The information in the two different databases is merged into the employee relation of the integrated environment by means of the W_DB1 and W_DB2 wrappers, respectively.
Mediator1 then defines a new employee relation that includes a user name (e.g. here it can be the name of the user who queries the integrated system), and it is defined by the set of employees "deducible" in the union of the two wrappers.
Mediator2 define the same employee relation given by the same relation in mediator1 "constrained" by the rules in the "Security Rules" module. The rule in SR serves, in this example, to limitate the access to the information on the employees of the whole integrated system. In this example only users that have a Level_C permission (user3 and user4), have access to this information.

5 CONCLUSIONS

Our research on deductive database systems and, in particular, on the problem of integrating different databases led us to the design of a language that supports the notion of mediation via a suite of operators for combining collection of clauses and the "in" feature.

The most interesting aspect of MedLan operators is the constraint (/) operator that allows for different applications (Asirelli*et al.* 1998) that we have studied and developed in the past and that are presently further investigated.

We paid special attention to a formal definition of the semantics of the language. The benefits of such an effort are that the operational and denotational semantics give us the way to implement a top-down and bottom-up evaluation of a query through the mediator which also give Medlan the capability to deal with "virtual" and "materialized" view approaches (Hull 1997)

One of the goal of this paper was the presentation of the semi-naive implementation of the language. This bottom-up implementation technique, that allows an efficient processing of universal queries, has been extended to queries involving Medlan program expressions.

We are currently studying different aspects of MedLan to extend it to cope with different integration mechanisms, such as cooperation and federation. Future work will concentrate on experimenting the characteristics of MedLan in various application domains. We find that the use of the constraint operator to deal with security issues is at the same time a demanding application field for MedLan and a promising solution.

In fact, with respect to other approaches we believe that our approach is novel in many respect. In particular, we believe that one of the interesting aspects of our approach is the idea of using the notion of "view" for implementing the control of accesses. In this way, instead of checking the user rights for accessing some information, we give the user its own "allowed" view on the database, in a very straightforward way.

Future work on the application we have presented will concern the study of different security models and enforcing policies existing in the literature to evaluate the feasibility of more complex examples and the effectiveness of our approach in real systems.

REFERENCES

Abitebul,S. Hull,R. and Vianu, V. (1995) *Foundations of Databases*, Addison Wesley.

Aquilino, D. Asirelli, P. Renso, C. and Turini, F. (1995) An Operator for Composing Deductive Databases with Theories of Constraints, in *Proc. of Logic programming and Non-Monotonic Reasoning '95, Lecture Notes in Computer Science*, **928**, Springer-Verlag, Berlin.

Aquilino, D. Asirelli, P. Renso, C. and Turini, F. (1997) Applying Restriction Constraints to Deductive Databases, In *Non-determinism in Deductive Databases*, (ed. D. Pedreschi and V.S. Subrahmaniam), *Annals of Mathematics and Artificial Intelligence*, **19(1,2)**, 3–25.

Aquilino, D. Asirelli, P. Formuso, A. Renso, C. and Turini, F. (1996), Using MedLan to Integrate Geographical Data , *accepted for publication on JLP.*

Asirelli, P. and Renso, C. (1997) The Constraint Operator in the MedLan Language, *Compulog Network area Constraint Programming and ERCIM Working Group on Constraints, Joint Workshop in conjunction with the CP97 Conference -Linz, Austria* , http://repl.iei.pi.cnr.it/people/asirelli/Publications/EWGC-2-97.html.

Asirelli, P. Renso, C. and Turini, F. (1996) Language Extensions for Semantic Integration of Deductive Databases, *Proc. of the International Workshop on Logic In Databases* (LID'96), *Lecture Notes in Computer Science*, **1154**, Springer-Verlag, Berlin.

Bertino, E. Jajodia, S. and Samarati, P. (1995) Database security: Research and practice, *Information Systems*, **20 (7)**, 537–556.

Bonatti, P. A. Kraus, S. and Subrahmanian, V. S. (1995) Foundations of Secure Deductive Databases, *IEEE Transaction on Knowledge and Data Engineering*, **7 (3)**, 406-422.

Brogi, A. Mancarella, P. Pedreschi, D. and Turini, F. (1994) Modular Logic Programming, *ACM Transactions on Programming Languages and Systems*, **16(4)**,1361–1398.

Brogi, A. (1993) Program Construction in Computational Logic *Ph.D. Thesis at University of Pisa.*

Brogi,A. Renso, C. and Turini, F. (1997) Dynamic Composition of Parameterized Logic Programs, "Submitted for publication"

Bertino, E. Samarati, P. and Jajodia, S. (1997) An Extended Authorization Model for Relational Databases, *IEEE Transaction on Knowledge and Data Engineering*, **9 (1)**,85–100.

Candan, K. S. Jajodia, S. and Subrahmanian, V. S. (1996) Secure Mediated Databases, *International Conference on Data Engineering 1996*, 28-37.

Jajodia, S. (1996a) Database security and privacy, *ACM Computing Surveys*, 50th anniversary commemorative issue, **28 (1)**.

Jajodia, S. (1996b) Managing Security and Privacy of Information, *ACM Computing Surveys* , **28 (4es)**, 129–131.

Jajodia, S. and Samarati, P., Subrahmanian, V.S. and Bertino, E. (1997) A Unified Framework for Enforcing Multiple Access Control Policies, *ACM SIGMOD.*

Hull R. (1997), Managing Semantic Heterogeneity in Databases: A Theoretical Perspective, Can be found in http://www-db.research.bell-labs.com/user/hull/pods97-tutorial.html

Lu, J. J. Moerkotte, G. Schue, J. and Subrahmanian, V. S. (1995) Efficient Maintenance of Materialized Mediated Views, *SIGMOD Conference 1995*, 340–351.

Papakostantinou, Y., Garcia-Molina, H. and Ullman, J. (1996) MedMaker: A mediation System Based on Declarative Specifications, *International Conference on Data Engineering.*

Renso, C. (1998) Mechanisms for Semantic Integration of Deductive Databases, *Ph.D. Thesis, University of Pisa, May 1998.*

Subrahmanian, V. S. (1994) Amalgamating Knowledge Bases, *Transaction On Database System*, **19(2)**, 291–331.

van Emden, M. H. and Kowalski,R. A. (1976) The semantics of predicate logic as a programming language, *Journal of the ACM*, **23(4)**,733–742.

Wiederhold G. (1992), Mediators in the Architecture of Future Information

Systems, *IEEE Computer*, **25**, 38–49.

Wiederhold, G., Bilello, M., Sarathy, V. and Qian XiaoLei (1996a) A Security Mediator for Health Care Information, *AMIA, Proc. of the 1996 AMIA Conf.*, 120–124.

Wiederhold, G., Bilello, M., Sarathy, V. and Qian XiaoLei (1996b) Protecting Collaboration, *Proc. of the NISSC'96.*

Ullman, J. (1988) *Principles of Database and Knowledge-Base Systems*, Computer Science Press.

6 BIOGRAPHY

Patrizia Asirelli graduated in Computer Science (Laurea in Scienze dell' Informazione) in 1975, from the University of Pisa. Since 1978 she is researcher working for the Italian National Research Council (CNR). She has been research group leader for many years and responsible for many research projects. She is the Secretary of the Italian Association for Logic Programming (GULP). Her research interests include logic programming, deductive databases and software engineering.

Chiara Renso graduated in Computer Science (Laurea in Scienze dell' Informazione) in 1992, from the University of Pisa. Got her PhD in Computer Science from the University of Pisa in May 1998 with a thesis on "Mechanisms for Semantic Integration of Deductive Databases" under the supervision of Prof. Franco Turini.

Franco Turini was born in 1949 in Italy. He graduated in Computer Science (Laurea in Scienze dell' Informazione) summa cum laude in 1973, from the University of Pisa. He is currently a full professor in the Department of Computer Science of the University of Pisa. In 78/80 he has been a visiting scientist of the Carnegie-Mellon University (Pittsburgh) and of the IBM Research Center S.Jose, afterwards. In 92/93 he has been visiting professor at the University of Utah. His research interests include programming languages design, implementation, and formal semantics especially in the field of functional and logic programming.