Design Principles for Secure Integration of Information

Jacques Calmet, Joachim Schü *
University of Karlsruhe
{calmet,schue}@ira.uka.de

Keywords: Mediators, Heterogeneous Databases, Security, Information Systems

1 Motivations

Over the last few years, the use of the Internet has been growing at a tremendous rate and access to different data- and knowledge sources provides a host of information from external and internal sources. This has led to at least three kinds of explosions:

- the number of *users* accessing the Internet has grown dramatically,
- the number of *sources of data* and software on the Internet has experienced a similar growth, and
- this had led to an equally impressive growth in the *heterogeneity* of databases, data formats, data structures and knowledge sources within which the above-mentioned users store their data/knowledge.

These are problems that must be taken into account when designing an intelligent decision support system. Indeed, they also partly or fully exist for local area networks (LAN). In our work, we have primarily studied a host of theoretical and practical issues arising out of the third point listed above [S93, LNS95, Sch95].

Wiederhold has proposed a layered mediator architecture [Wie92, Wie93] to provide end-user applications with information obtained through selection, abstraction, fusion, caching, extrapolation and pruning of data [Wie93]. The data is obtained from different independently developed data- and knowledge sources. In this ongoing project [1] we have developed techniques not only to facilitate the integration of different databases (object-oriented, relational and deductive), but also of arbitrary pieces of information, such as spreadsheets or even on-line information. The underlying theory of this project can be found in [S93, LNS95, Sch95]. In this paper, we show how techniques from artificial intelligence may provide the necessary

*Department of Computer Science, Institute for Algorithms and Cognitive Systems, University of Karlsruhe,76128 Karlsruhe, Am Fasanengarten 5, Tel: +49 721 608 4208, Fax: +49 721 608 6116

[1] In collaboration with the University of Maryland [LNS95, Sub95].

tools for building intelligent systems. In particular, the maintenance of inter-operable systems has been an expensive task which can be drastically reduced by employing some techniques from artificial intelligence, namely declarative programming.

In the long run we expect the emergence of *standards* for accessing remote data and in particular some widely accepted common data exchange formats such as GIF or ASCII proposed by large industrial consortium. Although, some of these standards for accessing relational and object-oriented databases such as Microsoft's ODBC or ODMG [Cat94] may provide a uniform application programming interface (API), the actual integration at the application software level still depends on the application programmer. Incorporating any new knowledge source requires still some procedural coding. However, the actual integration at the application software level can be drastically simplified by declarative programming techniques. Furthermore, queries spanning several data- and knowledge sources are not expressible with current database query languages. It is precisely the purpose of our ongoing project, to facilitate the integration of software by means of declarative programming as well as to provide a declarative query-language spanning multiple knowledge-sources.

The extension of classical logic programming to constraint logic programming (CLP) [JL87] has been a major theoretical and practical breakthrough. The most attractive aspect of the CLP scheme lies in the integration of the advantages of both declarative and imperative programming. On the one hand, CLP is a logical language, possessing similar declarative characterization as classical logic programming, thus enabling the *rapid development* of applications. On the other hand, CLP pushes many sub-deductions – especially those with well understood algorithmic solutions – into constraint domains. These can then be implemented efficiently using imperative languages.

The purpose of this paper is to twofold. Firstly, we will show how various conflicts which may arise in a mediated system can be handled by means of annotated constraint logic. Secondly, we will briefly discuss some security related issues in a mediator architecture and possible solutions which can be implemented within our system.

2 Preliminaries

In this section, we will sketch the basic theory behind a mediator knowledge base originally proposed in [LNS95, LMSS95, Sub94].

A *domain*, $\mathcal{D}$, is an abstraction of databases and software packages and consists of three components: (1) a set, Σ whose elements may be thought of as the data-objects that are being manipulated by the package in question, (2) a set $\mathcal{F}$ of functions on Σ. These functions take objects in Σ as input, and return, as output, objects from their range (which needs to be specified). The functions in $\mathcal{F}$ may be thought of as the predefined functions that have been implemented in the software package being considered, (3) a set of relations on the data-objects in Σ. Intuitively, these relations may be thought of as the predefined relations in the domain $\mathcal{D}$.

A *domain call* in our framework is a syntactic expression of the form

$$\texttt{domainname} :: \langle \texttt{domainfunction} \rangle (\langle \texttt{arg1}, \ldots, \texttt{argn} \rangle)$$

where `domainfunction` is the name of a function, and $\langle$`arg1`$,\ldots,$`argn`$\rangle$ are the arguments it takes. Intuitively, a domain call may be read as follows: in the domain called `domainname`, execute the function `domainfunction` defined therein on the arguments (`arg1`$,\ldots,$`argn`). The *result* of executing this domain call is coerced into a set of entities that have the same type as the output type of the function `domainfunction` on the arguments (`arg1`$,\ldots,$`argn`).

A *domain-call atom* (DCA-atom) is of the form `in(X,domainfunction`$\rangle$($\langle$`arg1`$,\ldots,$`argn`$\rangle$) where `in` is a constraint that is satisfied just in case the entity `X` is in the set returned by the domain call in the second argument of `in(X,Y)`. In other words, `in` is the polymorphic set membership predicate.

Furthermore, in contrast to [LNS95, Sub94] we introduce modes for constraint predicates and functions to ensure a correct instantiation of variables in the domain call atoms [CGH93]:

Mode	before	after
+	ground	ground
-	arbitrary	ground
?	arbitrary	arbitrary

"+" means that the argument must be ground before testing the constraint predicates/invoking constraint functions, "−" means that the argument must be ground after calling the external function, and "?" means that the variable instantiation is arbitrary. For example with $Student(+,+,-)$ and $Supervisor(-,+)$

$$\begin{aligned} Student(Phone) \leftarrow\ & Oracle :: Student(Name, Id, Phone), \\ & DBase :: Supervisor(SId, SName), \\ & SName = `JohnSmith', SId = Id \| \\ & Likes\_Soccer(Name) \end{aligned}$$

defines a predicate *Student* in the mediator which accesses two relational databases. A different mode for Supervisor would be $Supervisor(-,-)$ which would result in the same set of instances for Student. However, an important difference is that when testing a constraint containing the relation *Supervisor* for satisfiability, the *entire* relation *Supervisor* from the underlying Oracle database is materialized. *Afterwards* the constraint $SName = `JohnSmith'$ is being evaluated. Clearly, this would lead to a less efficient solution in contrast to querying the underlying database with the following SQL query:

```
select SId,SName
from Supervisor
where SName='John Smith'.
```

The introduction of modes is therefore very useful in fine tuning the access to the underlying knowledge sources.

The underlying knowledge sources are modeled in object-oriented manner: For instance, a constraint domain $D = Relational\_Database, F = select, R = \{Student, Supervisor\}$ may have subclasses $Oracle, DBase$ and $Paradox$. In our actual system there is a much more ramified taxonomy of knowledge sources, e.g. Object-oriented databases, Relational Databases, Relational Databases with trigger mechanisms, Spreadsheets and so on. As indicated, for

certain kinds of knowledge sources, the access mechanisms are similar or even *reusable* to a certain degree. For instance, most relational databases may be accessed via ODBC, an Excel Spreadsheet is accessible via OLE2. Another reason for polymorphism in the mediator is to simplify the formulation of queries.

The salient features of the language proposed in [LNS95, KS92, KE92] are the so-called *annotations* μ_i's which are constants, variables and terms over a complete lattice $\mathcal{T}$. Informally, annotations could be understood as the degree of *belief* in a proposition, e.g. $p : [0.5]$ means that the belief in the truth of A is *at least* $0.5 \in [0, 1]$. In [KS92] many different lattices have been proposed for reasoning with temporal, uncertain and inconsistent information.

Definition 2.1 (Annotated constrained clause) If $A : \rho$ is an annotated atom and $B_1 : \mu, \ldots, B_k$ are constant or variable annotated atoms, then

$$A : \rho \leftarrow B_1 : \mu_1 \wedge \ldots \wedge B_k : \mu_k$$

is an *annotated clause*. $A : \rho$ is called the head of this clause, whereas $B_1 : \mu_1, \ldots, B_k : \mu_k$ is called the *body*. All variables (object or annotation) are implicitly universally quantified. Any set of annotated clauses is also a GAP (Generalized Annotated Logic Programming).

A *mediator* is a set of rules of the form

$$A : \mu \quad \leftarrow \quad D_1 \wedge \ldots \wedge D_m || A_1 : \mu_1, \ldots, A_n : \mu_n$$

where $A : \mu, A_1 : \mu_1, \ldots, A_n : \mu_n$ are atoms, and $D_1, \ldots, D_m$ are DCA-atoms. Our mediator architecture allows not only to form queries by means of the above described languages, but also to implement some algorithmic solutions where different knowledge sources are being accessed. Furthermore, another very important feature of our mediator architecture is the ability to formulate integrity constraints spanning different independently developed knowledge sources. For instance, a clause may prohibit loans granted by some individuals without reconsulting some supervisor. This is, for instance, an important issue in a banking environment.

We work through the examples of [SK93] where possible conflicts in a heterogeneous environment have been classified.

3 Resolving conflicts

In [SK93] schematic and data conflicts have been classified and enumerated. In this section, we will work through this classification and show how it is possible to resolve the different conflicts within our framework.

3.1 Semantic Similarities between Objects

A relation $Professor(Id, Salary)$ in the domain Oracle and $Secretary(Id, Salary)$ in the domain Paradox may be *generalized* in the *context* of the university administration office:

$$\begin{aligned} Staff(Id, Name) : [\{m\}, \mathbf{t}] &\leftarrow Oracle :: Professor(Id, Name) \\ Staff(Id, Name) : [\{m\}, \mathbf{t}] &\leftarrow Paradox :: Secretary(Id, Name) \end{aligned}$$

Furthermore, the annotations of predicate symbols allow to express semantic proximity [SK93] be means of fuzzy values. For instance, the clauses

$$\begin{aligned} Student(Id, Name) : [0.8, \mathbf{t}] &\leftarrow Oracle :: Student_2(Id, Name, Grade) \\ Student(Id, Name) : [0.6, \mathbf{t}] &\leftarrow Paradox :: Student_1(Id, Name, Address) \end{aligned}$$

may express the semantic proximity between two objects from an Oracle and a Paradox Database. A more detailed description of semantic proximity and uncertainty modeling can be found in [SK93].

3.2 Domain Incompatibility

1. Synonyms: In two different knowledge sources the identity between the attribute Id and SId occurring in two unrelated relations

   $$\begin{aligned} Student(Id, Name, Address) \\ Teacher(SId, Name, Address) \end{aligned}$$

   in an Oracle and a DBase database may be established by the following clauses:

   $$\begin{aligned} Student\_Id(Id) : [\{m\}, \mathbf{t}] &\leftarrow Oracle :: Student(Id, Name) \\ Student\_Id(Id) : [\{m\}, \mathbf{t}] &\leftarrow Oracle :: Teacher(Name, SId), Name = `JohnSmith' \end{aligned}$$

   with $Oracle :: Student(-, ?)$ and $Oracle :: Teacher(+, -)$. It should be noted that the problem of homonyms does not occur in our approach since we assume all objects semantically unrelated. Also, this assumes that the result of the functions $Select$ returns the object which is representable in the data model of the mediator.

2. Data Representation Conflicts: Suppose that in the above example in the Oracle database student Id is only defined as a 9 digit integer which is directly representable in the data model of the mediator. The DBase select, where Teacher is defined as String, requires some data conversion by means of a system function $Convert\_Str\_to\_Int$:

   $$Student(Id) : [\{m\}, \mathbf{t}] \leftarrow Oracle :: (System :: Convert\_Str\_to\_Int(Id), Name)$$

   Similar functions may need to be provided for data scaling and data precision conflicts.

3. Default Value Conflicts: In two different knowledge sources there can be different default values for some related objects. Suppose that in one database the default value of driver's age is 18, while in another database the driver's age is 21. Although, it is not possible to map these different default values, we distinguish between the different knowledge sources by tagging the corresponding predicates in the mediator.

   $$\begin{aligned} LegalDriver(Name) : [\{1\}, \mathbf{t}] &\leftarrow Oracle_1 :: Driver(Age, Name), Age \geq 21 \\ LegalDriver(Name) : [\{2\}, \mathbf{t}] &\leftarrow Oracle_2 :: Driver(Age, Name), Age \geq 18 \end{aligned}$$

In the above clauses, 1 and 2 are ids from the lattice $(\mathcal{P}(\{Ag_1, \ldots, Ag_n\}), \subseteq)$ of the underlying knowledge sources [Sub94].

3.3 Entity Definition Incompatibility Problem

1. Database Identifier Conflict: Suppose that there are two different keys of two relations, such as $Student_1(Id, Course, Grade)$ and $Student_2(Name, Course, Grade)$, where Id and $Name$ are keys. If there is a mapping $convert_i$ from both keys to an abstract key, we may express the *semantic proximity* as

$$\begin{aligned} Student(Course, Grade) : [\mathbf{t}] \quad &\leftarrow \quad Oracle :: Student_1(System :: convert_1(ab\_key), \\ &\qquad Course, Grade) \\ Student(Course, Grade) : [\mathbf{t}] \quad &\leftarrow \quad Oracle :: Student_2(System :: convert_2(ab\_key), \\ &\qquad Course, Grade) \end{aligned}$$

   with modes $convert_i(+), Student_i(+, -, -)$.

2. Naming Conflict: In two different databases two entities Employee and Workers may denote the same set of entities. Although, this situation is different from the case of attribute naming conflicts, the same mechanism may ne used to map objects from Employee and Workers to the same predicate.

3. Union Compatibility Conflicts: Suppose that there is a DBase relational database containing the relation $Student(Id, Name, Grade)$ and furthermore an Oracle database containing the same relation $Student(Id, Name, Addess)$ denoting the same entities. Then, the following clauses may map them into a mediator predicate.

$$\begin{aligned} Student(Id, Name) : [\mathbf{t}] \quad &\leftarrow \quad DBase :: Student(Id, Name, Address) \\ Student(Id, Name) : [\mathbf{t}] \quad &\leftarrow \quad Oracle :: Student(Id, Name, Grade) \end{aligned}$$

   with modes $DBase :: Student(-, -, ?)$ and also $Oracle :: Student(-, -, ?)$.

4. Schema Isomorphism Conflicts: In this conflict, semantically similar entities have a different numbers of attributes.

$$\begin{aligned} &Instructor(No, Phone) : [\mathbf{t}] \leftarrow \\ &\quad Oracle :: Instructor(SS, HomePhone, OffPhone), Phone = HomePhone \\ &Instructor(No, Phone) : [\mathbf{t}] \leftarrow \\ &\quad Oracle :: Instructor(SS, HomePhone, OffPhone), Phone = OffPhone \end{aligned}$$

   with the mode $Instructor(-, -, -)$ and $- = +$.

5. Missing Data Item Conflict: Suppose that there are two relations and

$$Student(SS, Name, Type) \text{ and } Grad\_Student(SS, Name)$$

. The following clause maps $Grad\_Student$ to $Students$:

$$\begin{aligned} Student(SS, Name, Type) : [\mathbf{t}] &\leftarrow Oracle :: Student(SS, Name, Type) \\ Student(SS, Name, Type) : [\mathbf{t}] &\leftarrow DBase :: Grad\_Student(SS, Name), \\ &\quad Type = \text{`}Grad\text{'} \end{aligned}$$

with mode $DBase :: Grad\_Student(-,-)$ and $DBase :: Grad\_Student(-,-,-)$ and in particular $- = +$.

3.4 Data Value Incompatibility Problem

This class of conflicts arises due to different data values for the same attribute/object in a database. For instance a tuple $Salary(JohnSmith, 100k)$ in one database in contrast to $Salary(JohnSmith, 200k)$ in a different database. Resolving those kinds of conflicts is based upon the generalized annotated logic framework where different kinds of conflict solution strategies may be expressed.

- Known Inconsistency On this case the inconsistency is known and some kind of *priority ordering* may be specified in the mediator. For instance, the reliability of a particular knowledge source may provide such an ordering.

$$\begin{aligned} Salary(Name, Sal) : [\{1\}, \mathbf{t}] &\leftarrow Oracle :: Salary(Name, Sal) \\ Salary(Name, Sal) : [\{2\}, \mathbf{t}] &\leftarrow DBase :: Salary(Name, Sal) \\ Salary(Name, Sal) : [\{m\}, \mathbf{t}] &\leftarrow Sal \neq Sal2 || Salary(Name, Sal) : [\{1\}, \mathbf{t}], \\ &\quad Salary(Name, Sal2) : [\{2\}, \mathbf{t}] \end{aligned}$$

with modes $Oracle :: Salary(-,-)$ and $- \neq -$ and $- = -$.

- Temporary Inconsistency In this case one of the databases may have obsolete information.

$$\begin{aligned} Salary(Name, Sal) : [\mathbf{t}, \{Mo, Tue\}] &\leftarrow Oracle :: Salary(Name, Sal) \\ Salary(Name, Sal) : [\mathbf{t}, \{Fri, Sat\}] &\leftarrow DBase :: Salary(Name, Sal) \end{aligned}$$

Since annotated logic is able to mimic different kinds of temporal reasoning, much more sophisticated ways of resolving temporary inconsistency are expressible in our mediator architecture. For more details on temporal logic programming with annotated logic, the reader may refer to [KS92, LNS95].

- Acceptable Inconsistency In certain cases one would like to tolerate certain degrees of inconsistency:

$$\begin{aligned} Salary(Name, Sal) : [\{m\}, \mathbf{t}] &\leftarrow Abs(Sal - Sal2) \leq 10 \,|| \\ &\quad Salary(Name, Sal) : [\{1\}, \mathbf{t}], \\ &\quad Salary(Name, Sal2) : [\{2\}, \mathbf{t}] \end{aligned}$$

3.5 Abstraction Level Incompatibility Problem

Consider the following example, where in three different databases the relations are defined as follows:

DB1

Date	Company	Price
27-09-95	BMW	10.30
26-09-95	Daimler	89.32

DB2

Date	BMW	Daimler
27-09-95	10.30	8.30
26-09-95	78.40	89.32

DB3

Date	BMW	Date	Daimler
27-09-95	10.30	27-09-95	89.32

There are several kinds of conflicts which may be resolved as follows:

$$
\begin{aligned}
Stock(Date, Price, Company) : [\mathbf{t}] \quad &\leftarrow \quad DB1 :: Stock(Date, Company, Price) \\
Stock(Date, Price, Company) : [\mathbf{t}] \quad &\leftarrow \quad DB2 :: Stock(Date, BMW, Daimler), \\
&\qquad\quad Price = BMW, \\
&\qquad\quad Company = \text{`}BMW' \\
Stock(Date, Price, Company) : [\mathbf{t}] \quad &\leftarrow \quad DB2 :: Stock(Date, BMW, Daimler), \\
&\qquad\quad Price = Daimler, Company = \text{`}Daimler' \\
Stock(Date, Price, Company) : [\mathbf{t}] \quad &\leftarrow \quad DB3 :: Stock(Date, BMW), \\
&\qquad\quad Price = BMW, Company = \text{`}BMW' \\
Stock(Date, Price, Company) : [\mathbf{t}] \quad &\leftarrow \quad DB3 :: Stock(Date, Daimler), \\
&\qquad\quad Price = Daimler, Company = \text{`}Daimler'
\end{aligned}
$$

Alternatively, instead of coding these transformations into the logic, one may write some C-Code function which takes as input a relation from the constraint domain and outputs the relation transformed into the common data model of the mediator.

4 Security in a Mediator Architecture

Security in a mediator architecture is quite similar to security within federated databases. See for instance [Per93, CJS95] for relevant references. Security basically comprises two main features: confidentiality and integrity.

- Confidentiality is enforced in two steps. In the initial step, the identification of accessing users and the authentification of their credentials through an audit is performed. The second step consists in verifying that the identified users have been granted permission to access the underlying software packages.

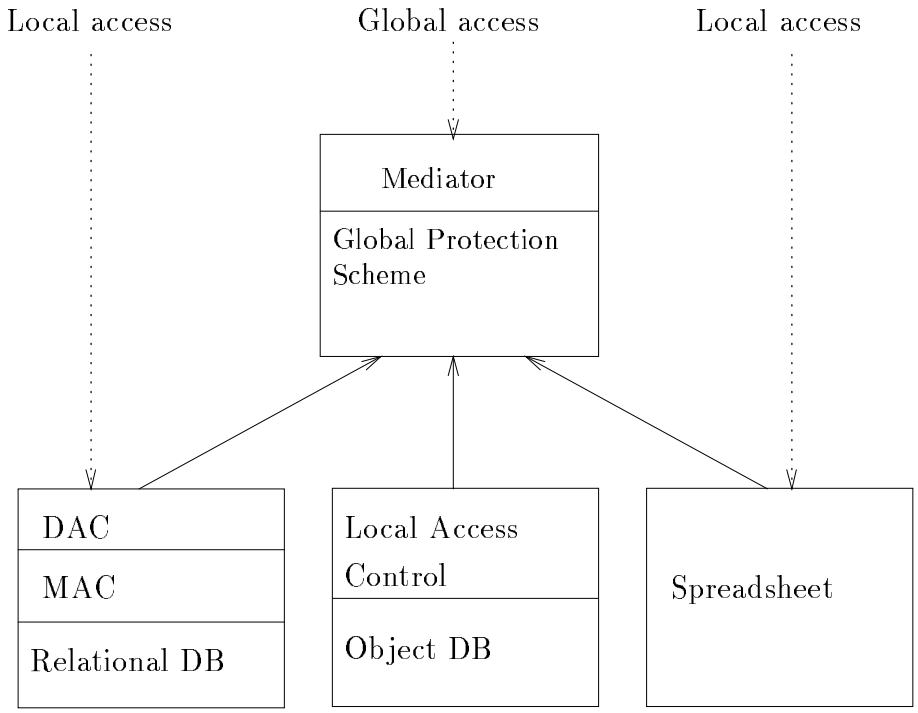

Figure 1: Security in a mediator architecture

- Integrity requires to monitor any change in the knowledge bases and to monitor their consistency.

In most commercial database discretionary access control (DAC) and mandantory access control (MAC) are implemented. While the first basically filters all the data a user is not authorized to access by means of views, MAC deals with flow of information. In [Per93] a common security model which is able to mimic DAC as well as MAC based local security policies has been proposed (see also figure 1).

So far we have described and illustrated that our approach enforces the semantical consistency of knowledge bases and thus, most probably, enables to investigate this problem. We are now concerned only with the confidentiality aspect of security. Our starting ideas are twofold. Firstly, confidential access to knowledge bases is no different from confidentiality in a network of computers. Secondly, it is not sufficient to enforce confidentiality at access time. Indeed, confidentiality must be preserved at the level of any transaction in the mediator. To illustrate such a need consider two examples. In the first one, an intruder managed to break through the authentification and authorization controls. He can then navigate freely within the system without being further controlled. In a second example, a user is authorized to access a system of federated databases but a supervisor wishes to set a limit on either the number or on the contents of allowed transactions. This is however presently almost impossible to enforce, although this would have been useful in the celebrated case of the Barings bank for instance. Another requirement is that security be achieved through a zero-knowledge identification mechanism [BFE92]. Our solutions to these problems are based upon the fact

that we have clauses with annotations in a mediatory system. We give only a very brief sketch of these ideas. The security mechanism is based upon SELANE (see [BGD93] for a description and relevant definitions and references) that stands for Secure Local Area Network Environment and has been designed and implemented in our Institute. It proposes the use of secure key issuing authorities (SKIA), which provide principals with certificates required for one way or mutual authentification. Each principal's certificate contains a long term public and private key. The so-called ElGamal public key signature scheme is used for user registration and authentification. The one-way function used in this scheme is an exponentiation modulo an appropriate number p. Authentification is achieved by establishing a common session key that is derived from the certificates issued by the SKIA. This key can be used during transactions to ensure confidentiality. In our context, the exponential is used as an annotation in the clauses of GAP and the security scheme is managed by the mediator. The lattice associated to this function is simply made of the true and false values whether authentification and certification are granted or not. This is checked through the mechanisms described in section 3.2. Such an approach allows also to issue certificates at the level of any transaction in the system. In addition, the supervisor/mediator can issue "annotations" that limit the range of allowed transactions.

5 Conclusion

We have demonstrated how any kind of software can be integrated in a declarative manner. Most recent attempts only integrated relational and object-oriented databases. The resulting system is implemented and ready for use in various application domains. We have also underlined how a security mechanism, based on existing methods, can be extended to mediated knowledge bases. This is however not yet implemented. The results sketched in this paper are applicable to the design of secure decision support systems in a straightforward manner.

References

[BFE92] Th. Beth, M. Frisch, and G.J. Simons (Eds.). *Public-Key Cryptography: State of the Art and Future Directions.* Springer LNCS 578, 1992.

[BGD93] Th. Beth, D. Gollmann, and F. Damm. Authentification services in distributed systems. *Computer and Security*, 12:753–764, 1993.

[Cat94] R. Cattell. *The Object Database Standard: ODMG-93.* Morgan Kaufmann Publishers, San Mateo, California, 1994.

[CGH93] A. B. Cremers, U. Griefahn, and R. Hinze. *Deductive Databases (in german).* Vieweg, 1993.

[CJS95] K.S. Candan, S. Jajodia, and V.S. Subrahmanian. Secure mediated databases. Draft, University of Maryland, 1995.

[JL87] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *Proceedings of Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, 1987.

[KE92] M. Kifer and L. Lozinskii E.˙ A logic for reasoning with inconsistency. *Journal of Automated Reasoning*, 9:179–215, 1992.

[KS92] M. Kifer and V.S. Subrahmanian. Theory of Generalized Annotated Logic Programming. *Journal of Logic Programming*, 12(1):335–367, 1992.

[LMSS95] J. Lu, G. Moerkotte, J. Schü, and V.S. Subrahmanian. Efficient maintenance of materialized mediated views. In *Proceedings of the ACM SIGMOD Annual Conference*, San Jose, CA, May 1995. ACM Press.

[LNS95] J. Lu, A. Nerode, and V.S. Subrahmanian. Hybrid knowledge bases. *To appear in IEEE Transactions on Data and Knowledge Engineering*, 1995.

[Per93] G. Pernul. Canonical security modeling for federated databases. In *Interoperable Database Systems*, pages 207–222, 1993.

[S93] S. Adalı and V.S. Subrahmanian. Amalgamating knowledge bases, iii: Algorithms, data structures and query processing. To appear in Journal of Logic Programming, 1993.

[Sch95] J. Schü. *Updates and Query-Processing in a Mediator Architecture.* PhD thesis, University of Karlsruhe, October 1995.

[SK93] A. Sheth and V. Kashyap. So far (schematically) yet so near (semantically). In D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, editors, *Interoperable Database Systems*, pages 283–312. IFIP, Elsevier Science Pubslihers, 1993.

[Sub94] V.S. Subrahmanian. Amalgamating knowledge bases. *ACM Transactions on Database Systems*, 19(2):291–331, 1994.

[Sub95] V.S. Subrahmanian. Hermes: A heterogeneous reasoning and mediator system. University of Maryland, College Park, Draft, 1995.

[Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, march 1992.

[Wie93] G. Wiederhold. Intelligent integration of information. In *ACM SIGMOD Conference*, pages 434–437, May 1993.