TOWARDS THE DESIGN AND IMPLEMENTATION OF A MULTILEVEL SECURE DEDUCTIVE DATABASE MANAGEMENT SYSTEM

Bhavani Thuraisingham

The MITRE Corporation, Burlington Road, Bedford, MA 01730

ABSTRACT

In this paper we describe a preliminary design and implementation of a multilevel secure deductive database management system (MLS/DEDDBMS). In particular, logic as a data model for multilevel databases, reasoning across security levels, architectural issues for an MLS/DEDDBMS, and a prototype implementation are discussed.

1. INTRODUCTION

While relational database management systems have been adequate for present-day applications, complex operations of the future (for applications such as C3I, multimedia, AI, and process control) would require the inferencing power of deductive database management systems (DEDBMS). Such systems can be regarded as DBMSs with deduction capability. DEDDBMSs will enable (i) the efficient representation, storage, and management of complex applications, and (ii) the intelligent processing of the structures representing the complex applications.

Two of the approaches to developing a DEDBMS are: (i) loose coupling between a DBMS and a deductive manager, and (ii) tight coupling between a DBMS and a deductive manager. In the first approach, there is a clear separation between a DBMS, which is responsible for managing the database, and a deductive manager which is responsible for intelligent reasoning. In the second approach, the deductive manager is embedded into the DBMS [BRODU].

However, the increasing popularity of DEDBMSs should not obscure the need to maintain security of operation. That is, it is important that such systems operate securely in order to overcome any malicious corruption of data as well as to prohibit unauthorized access to and use of classified data. For many applications, it is also important to provide multilevel security. Consequently, multilevel secure deductive database management systems (MLS/DEDDBMSs) are needed in order to ensure that users cleared to different security levels access and share a database with data classified at different sensitivity levels in such a way that they obtain only the data classified at or below their level.

In this paper we describe a preliminary design and implementation of an MLS/DEDDBMS. We define a MLS/DEDDBMS to be multilevel secure relational DBMS (MLS/RDBMS) which is augmented by a deductive manager. The deductive manager is based on a multilevel secure logic programming system (MLS/LPS). That is, a loose-coupling approach between a DBMS and a deductive manager is chosen in our design of a MLS/DEDDBMS. Our architecture is illustrated in figure 1. As can be seen, user's query is posed in a logical language. The MLS/LPS processes the query from the information in the multilevel knowledge base. Whenever data is required from the multilevel database, appropriate requests are formulated and sent to the MLS/RDBMS. The response from the MLS/RDBMS is assembled with the response produced by the MLS/LPS and delivered to the user.

Figure 1. Architecture of an MLS/DEDDBMS
The organization of this paper is as follows. In section 2, we provide an overview of MLS/RDBMSs. In section 3, we discuss the design of an MLS/LPS. In particular, issues on logic as a data model for a multilevel environment, reasoning in a multilevel environment, as well as architectural issues are described. The prototype implementation is described in section 4. The paper is concluded in section 5.

2. OVERVIEW OF MLS/RDBMS

In an MLS/RDBMS, users who are cleared at different security levels access and share a database consisting of data at different sensitivity levels. The security levels may be assigned to the data depending on content, context, aggregation and time. It is generally assumed that the set of security levels forms a partially ordered lattice with Unclassified < Confidential < Secret < Top Secret. An effective security policy for an MLS/RDBMS should ensure that users only acquire the information at or below their level. The earliest of security policies, the Bell and LaPadula security model [BELL75], is stated below:

1. Subjects are the active entities (such as processes), and objects are the passive entities (such as files, relations, and tuples).
2. Subjects and objects are assigned security levels.
3. Simple security property: A subject has read access to an object if the subject's security level dominates the security level of the object.
4. Property: A subject has write access to an object if the subject's security level is dominated by the security level of the object.

The relational data model has dominated much of the work on multilevel secure DBMSs. Various extensions to the relational data model have been proposed in order to obtain a multilevel relational data model. We define a multilevel relation to be a relation in which each tuple is assigned a security level. Figure 1 shows a multilevel relation EMP. Note that the tuples in EMP are either Secret or Unclassified.

Furthermore, if SS# is taken to be the primary key of EMP, then we see that there are two tuples with the same primary key at different security levels. This feature is known as polyninstantiation. It is a mechanism that is used to handle cover stores in a multilevel world. That is, in a multilevel world, users at different levels could have different views of the same entity.

<table>
<thead>
<tr>
<th>SS#</th>
<th>Name</th>
<th>Salary</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>20 U</td>
<td>U</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
<td>30 U</td>
<td>U</td>
</tr>
<tr>
<td>3</td>
<td>James</td>
<td>40 U</td>
<td>U</td>
</tr>
<tr>
<td>4</td>
<td>John</td>
<td>70 S</td>
<td>S</td>
</tr>
<tr>
<td>5</td>
<td>Mary</td>
<td>80 S</td>
<td>S</td>
</tr>
<tr>
<td>6</td>
<td>James</td>
<td>60 S</td>
<td>S</td>
</tr>
</tbody>
</table>

Figure 2. Multilevel Relation

We assume that the security level of the schema of a relation is the security level of the user who creates the schema. The security level of tuple in a relation R should dominate the security level of the schema for R. That is, if the schema for the relation EMP is Unclassified, then there could be tuples in EMP at the Unclassified, Confidential, Secret, and Top Secret levels. A user can read any tuple whose security level is dominated by his level. A user can update a tuple classified at his level. That is, we do not permit the write-up property of the Bell and LaPadula security policy. If the user's security level is dominated by the security level of the schema of the relation, then the tuple is not entered.

Two of the architectures that have been proposed for an MLS/RDBMS are: (i) the operating system provides all mandatory access control, and (ii)

---

2. The terms security level and sensitivity level, will be used interchangeably. Some have also used the term classification level to mean security level.

3. Note that most MLS/RDBMSs enforce a restricted form of the *-property where a subject writes into an object if the subject's security level is that of the object.

4. Note the following definitions. Mandatory security controls restrict access to data depending on the sensitivity levels of the data and the clearance level of the user. By a sensitivity level, we mean the information attached to the data which reflects its sensitivity. Each user is cleared at a particular level which is called his clearance level. In contrast, discretionary security measures are usually in the form of rules which specify the type of access that users or groups of users may have to different kinds of data. A multilevel database is a database in which the data are assigned different sensitivity levels. For a discussion of these definitions, we refer the reader to [TCSEC85]. In this paper we are concerned only with mandatory access control.
the trusted subject-based architecture. In the first architecture, the operating system controls all mandatory access control to the objects. That is, the DBMS is untrusted with respect to mandatory access control. In this approach, a multilevel relation is decomposed and stored in single-level fragments. The tuples at level $L$ are stored in a fragment at level $L$. The operating system controls access to the fragments. Figure 3 shows how the relation EMP could be decomposed into two relations, EMP-U and EMP-S. EMP-U has all of the Unclassified tuples and EMP-S has all of the Secret tuples. EMP-U could be stored in an Unclassified file while EMP-S could be stored in a Secret file.

In the second architecture, the operating system provides access control to its files, but the DBMS is trusted to provide access control to its own objects such as tuples. In the example that we have considered, the multilevel relation of figure 2 will be stored in a Secret file. The DBMS which is trusted will filter out the Secret tuples if the request is from an Unclassified user. It must be ensured that untrusted subjects do not change the security labels (such as S, U, etc.) of the tuples stored in the Secret file.

<table>
<thead>
<tr>
<th>EMP-U</th>
<th>EMP-S</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS#</td>
<td>Name</td>
</tr>
<tr>
<td>1</td>
<td>John</td>
</tr>
<tr>
<td>2</td>
<td>Paul</td>
</tr>
<tr>
<td>3</td>
<td>James</td>
</tr>
</tbody>
</table>

Figure 3. Single-level Relations

Both architectures have advantages and disadvantages. In the first architecture, since the DBMS is untrusted, it is easier to evaluate such a system. However performance is an issue as the relations have to be decomposed and later recombined. In the second architecture, while performance is an advantage, since the DBMS is trusted, evaluation may be more complex.

A more detailed discussion on MLS/RDBMSs can be found in [AFSB83, DENN87, GARV88, STAC90].

The MLS/RDBMS that we have used in the implementation is based on the trusted subject-based architecture. We used such a system as it was available for our use. The design of the MLS/RDBMS is independent of the logic programming system that we have designed. Note that the logic programming system could also be designed using one of the two architectures described above. However, we have chosen the trusted subject-based architecture so that in addition to making deductions, the system could be utilized to handle certain security violations via inference. The architectural issues are discussed in section 3.

3. MULTILEVEL SECURE LOGIC PROGRAMMING SYSTEM

3.1 Overview

As stated in section 1, the deductive manager that augments the MLS/RDBMS is based on a logic programming system. Current logic programming systems are not adequate for multilevel secure applications. This is because current logic programming systems cannot reason across security levels. Therefore, they require the programmer to specify some of the control, thus causing it to lose the main advantages of a logic programming system. Therefore, we have designed a logic programming system to function in a multilevel environment. Our logic programming system is based on a logic that we have developed called NTML (Nonmonotonic Typed Multilevel Logic). NTML is based on first order logic with constructs to support multilevel security. It is nonmonotonic because information that is true at one level may be false at another level.

Note that the problem of deducing unauthorized information from legitimate information has come to be known as the inference problem. The inference problem is described in detail in [THUR91].

For a discussion on logic programming, we refer to [LLOY87].

A detailed discussion of NTML is given in [THUR90]. A version was also published in the proceedings of the the 4th IEEE Computer Security Foundations Workshop in Franconia, NH, June 1991. Note that in our discussion of NTML we assume that the security levels are strictly hierarchical. Extensions are needed to support nonhierarchical levels.
In NTML, each formula is assigned a security level and has a truth value at all levels which dominate its security level. For example, if a formula \( F \) is assigned the Confidential level, then it has truth values at the Confidential, Secret, and TopSecret levels. It has no truth value at the Unclassified level as it is assumed that the formula does not exist at the Unclassified level. In addition to the usual rules of first order logic, NTML has a special rule called DASL (Deduction Across Security Levels). It states that if a formula \( F \) is true at level \( L \), then it has truth value for all levels \( L^* > L \) provided the truth value does not change at a level \( L_1 \) where \( L < L_1 < L^* \). This means that if a formula \( F \) is true at the Confidential level and is false at the Secret level, then it is assumed to be false at the TopSecret level provided it is not explicitly asserted that the formula is true at the TopSecret level. That is, whenever there is a contradiction, information at the higher level is assumed to be true.

Figure 4 illustrates these points. Consider the formula \( \forall x P(x) \). Assuming that the predicate \( P \) is assigned the Unclassified level, the formula has truth values at each level which dominates the Unclassified level. Figure 4 illustrates the domains at different security levels (we have not considered the Confidential level in this example). At the Unclassified level, the domain has elements \( a_1, a_2, a_3 \). At the Secret level, the domain has elements \( a_1, a_2, a_3, a_4, a_5, a_6, a_7 \). \( P(x) \) is asserted true for \( a_1, a_2, a_3 \) and \( a_7 \). \( P(x) \) is asserted true for \( a_1, a_2, a_3, a_4, a_5, a_6, a_7 \). \( P(x) \) is asserted false for \( a_2 \) at the Secret level. Since nothing is specified for \( a_1 \) and \( a_3 \), it is assumed that \( P(x) \) is true for \( a_1 \) and \( a_3 \) at the Secret level. \( P(x) \) is asserted true for \( a_4 \) and \( a_5 \) at the Secret level. At the TopSecret level \( P(x) \) is asserted true for \( a_1 \). In addition, \( P(x) \) is asserted true for \( a_6 \) and \( a_7 \). Since nothing is specified for \( a_1, a_3, a_4, a_5 \), at the TopSecret levels, it is assumed that \( P(x) \) is true for these elements at the TopSecret level. The end result is that \( \forall x P(x) \) is true at both the Unclassified and TopSecret levels, but it is false at the Secret level.

The relationship between MLS/RDBMSs and NTML is analogous to the relationship between relational DBMSs and first order logic. Note that there are essentially three ways to view relational databases through logic [GALL78]. In the first approach, called the proof-theoretic approach, a database is viewed as a first order theory. Queries and integrity constraints (which are also called general laws) are expressed as formulas which have to be proved as theorems of the theory. Theorem proving techniques are used to evaluate queries as well as to process integrity constraints. The integrity constraints in this approach are called derivation rules. In the second approach, called the model theoretic approach, the database is regarded as an interpretation of a first order theory. Queries and integrity constraints are expressed as formulas of a first order theory which must be evaluated on the interpretation using the semantic definition of truth. Techniques from model theory are used in this evaluation. The integrity constraints in this approach are called integrity rules. A third approach, which is a combination of the first two approaches, has also been proposed. In this approach, called the integrated approach, some of the integrity constraints are treated as derivation rules and some as integrity rules. The database consists of an extensional portion and an intensional portion. The extensional portion of the database is a model of the integrity rules. The derivation rules are used to deduce implicit information.

Figure 4. Illustrating Deduction Across Security Levels

It is the third approach that has received much attention by the database community. Based on this approach, deductive database management systems, which extend relational database systems with deductive managers have been developed. The deductive managers, which are usually based on logic programming systems, are responsible for making deductions from existing information. We have taken the integrated approach to designing MLS/DEDDBMSs where the logic programming system is based on NTML. In section 3.2...
we describe the integrated approach. In section 3.3 we describe the reasoning procedure used by the logic programing system that we have designed. Architectural issues are described in section 3.4.

3.2 Viewing Multilevel Databases Through Logic

In the integrated approach, some of the general laws are taken as integrity rules and the others as derivation rules. The multilevel database has two components. One is the explicit extension which is a model of an NTLML theory whose proper axioms are the general laws which are regarded as integrity rules. The other is the implicit extension which is the set of all tuples which are derived from the explicit extension by virtue of the general laws which are used as derivation rules. Figure 5 illustrates this approach.11

Query Evaluation

Query evaluation at a level L depends on whether relations are defined explicitly or implicitly. Two ways to treat relation definitions have been identified. They are as follows:

(a) A Relation is either defined explicitly or implicitly. If it is defined implicitly, then it is defined in terms of the explicit relations. Note that each definition is a NTLML formula and therefore has a security level attached to it. When a query is posed at level L, all references to implicit relations in the query formula are replaced by explicit relations according to the definitions of the implicit relations at level L. The modified query is evaluated against the explicit extension of the multilevel database. Note that the treatment of views in relational database systems takes this approach.

(b) In the second approach, it is not necessarily the case that a relation is defined either implicitly or explicitly. That is, it is possible for part of the relation to be defined explicitly and part of it implicitly. Therefore, evaluating a query at level L amounts to deducing facts which are tuples at level L. That is, evaluating a query R(x1, x2, x3, ..., xn), expressed by (R(x1, x2, ..., xn), L), amounts to obtaining all proofs of the formula (E x1/t1, x2/t2, ..., xn/tn R(x1, x2, ..., xn), L).

An advantage of the second treatment of relation definition is that in the real world one could know various relationships without having to know exactly how these relationships were formed. That is, it is possible for people to be senior employees without actually having to make a salary of more than 30K.

---

11 We have adapted the work of [NICO78] for a multilevel environment.

---

Figure 5. Illustrating the Integrated Approach

Figure 6. A Multilevel Deductive Database

We illustrate query evaluation with an example. Consider the multilevel deductive database illustrated in figure 6. S1 and S2 are schema rules (we assume that SL is the set of security levels), DR1 is a derivation rule which defines senior employees. IR1, IR2, IR3, and IR4 are integrity rules. IR1 states at the Unclassified level that the salary values must be less than 40K. IR2 enforces the entity integrity property...
within a security level. That is, within a security level, no two tuples have the same primary key. IR3 negates IR1 at the Secret level. That is, at the Secret level employees may have salaries more than 40K. IR4 states that if a tuple exists at a particular security level and there is no tuple at a higher level with the same primary key, then the tuple at the lower security level is assumed to be true at the higher level (note that Just-less-than (L1, L2) means that there is no security level between L1 and L2 and L1 < L2). Because IR4 is used as an integrity rule, the tuples are duplicated at different security levels. Had IR4 been used as a derivation rule, then the tuples (111, James, 35K, Secret) and (222, Mary, 15K, Secret) need not be entered into the relation EMP. These tuples could have been derived by IR4.

Suppose a Secret user poses a query to retrieve all senior employees. This query is expressed by the NTML formula:

\[ \text{SEN-EMP}(X), \text{Secret}. \]

The answer to the query is the set \{John, James, Jane, Jill, David\}. John, Jill, and David are taken from the relation SEN-EMP explicitly defined. Jane and James are derived from the rule DR1.

Advantages and Drawbacks

By using some of the general laws as derivation rules, not all of the information has to be made explicit. This could save storage space. Also, with this approach it is possible to express general information about the perceived world without having to split it into sets of elementary information.

Negative information is handled by assuming that anything that is not either explicitly or implicitly specified in the database is assumed to be false. However, one has to be careful so as to not introduce wrong information. That is, one has to forbid the derivation of positive information from negative information as negative information can be regarded as information that is not positively sure. That is, the following general law \((G)\) cannot be used as derivation rule:

\[ (G): (A1 \land A2 \land A3 \land \ldots \land A_n \rightarrow B_1 \lor B_2 \lor B_3 \ldots \lor B_p), L; \text{ where } p > 1. \]

Note that the above law can be rewritten as:

\[ (A1 \land A2 \land A3 \land \ldots \land A_n \rightarrow B_1 \land \neg B_2 \land \neg B_3 \ldots \land \neg B_p), L. \]

The negative information \(\neg B_1, \neg B_2, \neg B_3 \ldots \neg B_p\) is included in the derivation of \(B_p\). Since this negative information could be obtained by default, (i.e.,

the negative information could be obtained by the absence of positive information), one cannot be completely certain of the validity of the negative information. As a result, one cannot be certain of the validity of any information derived from negative information. If \(p = 1\) in the general law \((G)\), then all information is deduced from positive information.

3.3. Reasoning Across Security Levels

In this section we describe the procedure that we have used for reasoning across security levels. As stated earlier, our logic is based on the special rule deduction across security levels. That is, if \(A\) is true at level \(L\) then \(A\) is assumed to be true at all levels \(L^* \geq L\) provided \(A\) is not negated at a level \(L_1\) where \(L < L_1 \leq L^*\). The reasoning procedure is implemented in the form of the resolution rule for a logic programming system for a multilevel environment. That is, we have designed a logic programming system which incorporates constructs for multilevel security.

A logic program for a multilevel environment consists of a set of program clauses. A program clause is of the form:

\[ (\neg (A,L) \leftarrow (B_1, L_1), (B_2, L_2), \ldots, (B_n, L_n)), L^* \]

where \(n \geq 0\) (note that if \(n = 0\), then there are no \(B_1\)s), \(L, L_1, L_2, \ldots, L_n, L^*\) are security levels \(L_1, L_2, \ldots, L_n\) are all \(\leq L^*\) (if the security levels are constants) \(A, B_1, B_2, \ldots, B_n\) are (positive) atomic NTML formulas.

The second form is the negation of the first form. The first form is read as follows:

"To show that \(A\) is true at level \(L\), we need to show that \(B_1, B_2, \ldots, B_n\) are all true at levels \(L_1, L_2, \ldots, L_n\), respectively. Further, this entire statement is true at the level \(L^*\)."

A goal clause is of the form:

\[ \leftarrow (B_1, L), (B_2, L), \ldots, (B_n, L) \]

where \(n \geq 1\), \(L_1, L_2, \ldots, L_n\) are security levels and \(B_1, B_2, \ldots, B_n\) are (positive) atomic NTML formulas.

Next, we describe the algorithm which implements the resolution rule for the system.
A query posed by a user at security level L is expressed as a goal associated with level L. Let the goal be

\[ \langle A_1, L \rangle, \langle A_2, L \rangle, \ldots, \langle A_m, L \rangle. \]

The resolution principle is implemented as follows.

For each \( i \) (1 \( \leq \) \( i \) \( \leq \) \( m \)) do the following:

- Unify \( \langle A_i, L \rangle \) with some program clause
- \( (((A_i \cdot L')) \leftarrow (B_1, L_1), (B_2, L_2), \ldots, (B_n, L_n)), L'') \)

where the following conditions are satisfied:

(a) \( L' \leq L \) if \( L' \) is a constant.
(b) It is not the case that there is a security level \( L'' \) where \( L' < L'' \leq L \) such that

\[ (((NOT ((A_i \cdot L')) \leftarrow (B_1, L_1), (B_2, L_2), \ldots, (B_n, L_n))), L'') \]

Note, by unification we mean find the most general unifier for \( A_i \) and \( A'' \) which satisfies the conditions stated above.

If a unifier cannot be found, then there is no solution for the query. Return "failure" as the response.

If there is such a unifier, then it is included in the response being assembled. Try the procedure (i.e., unification) for the \( \langle A_{i+1}, L \rangle \).

End (for each \( i \)).

Return the response to the query. If another solution has to be found, then repeat the same procedure, but exclude the previous responses in the unification process. That is, find only the new solutions.

This ends the algorithm for implementing the resolution principle.

The resolution principle as described here will ensure that appropriate responses at or below the user's level will be retrieved.

We illustrate this point with a simple example. Suppose we have the following clauses:

\[
\begin{align*}
C_1: & \quad (((\text{Works}(e_1,d_1), U) \leftarrow), U) \\
C_2: & \quad (((\text{Works}(e_2, d_1), S) \leftarrow), S) \\
C_3: & \quad (((\text{Works}(e_3, d_2), S) \leftarrow), S) \\
C_4: & \quad (((\text{Works}(x, z), L) \leftarrow \text{Works}(x,y), L) \\
& \quad \land \text{SubDept}(z,y), L)), U) \\
C_5: & \quad \text{NOT} (((\text{Works}(x,z), L) \leftarrow \text{Works}(x,y), L) \\
& \quad \land \text{SubDept}(z,y), L)), S) \\
C_6: & \quad (((\text{SubDept}(d_2,d_1), U) \leftarrow), U)
\end{align*}
\]

Clause C1 states that \( e_1 \) works in department \( d_1 \) and is classified at the Unclassified level, clause C2 states that \( e_2 \) works in department \( d_1 \) at the Secret level, clause C3 states that \( e_3 \) works in department \( d_2 \) at the Secret level, clause C4 states that if \( y \) is a subdepartment of \( z \) and if \( x \) works in \( y \) then \( x \) works in \( z \) at the Unclassified level, clause C5 negates clause C4 at the Secret level, and clause C6 states that \( d_1 \) is a subdepartment of \( d_2 \) at the Unclassified level.

Suppose an Unclassified user queries for those who work in department \( d_2 \). The query is expressed in the form \( \text{Works}(x,d_2), \text{Unclassified} \). Then the clauses C1, C4, and C6 will be used in the resolution process and the response will be \( e_1 \). If the same query is posed at the Secret level, then since clause C5 is the negation of clause C4, \( e_1 \) and \( e_2 \) will not be included in the response. The response to the query will then be just \( e_3 \).

Note that if the multilevel database contains the relation "Works", then the query will be issues to the MLS/RDBMS at the user's level. The response from the MLS/RDBMS will be assembled with the response produced by the MLS/LPS and the combined response will be delivered to the user.

### 3.4 Architectural Issues

In this section we discuss architectural issues for an MLS/LPS. As in the case of an MLS/RDBMS, both the OS-provides mandatory access control as well as the trusted subject architecture could be utilized for such a system. In the OS provided mandatory access control, one could envisage having a deductive manager running at each security level. The clauses are partitioned across security levels. That is, a clause classified at level \( L \) is stored in a file at level \( L \). The operating system provides access control to the clauses. That is the operating system will ensure that a subject can read a clause only if the subject's security level dominates the security level of the clause. The
MLS/LPS runs as an untrusted subject. When a user issues a query, the MLS/LPS will execute at the user’s level. It will have read access to all files at or below the user’s level. It will combine the appropriate clauses and implement the resolution procedure discussed in section 3.3. In the example that we considered previously, clauses C1, C4, and C6 may be stored in an Unclassified file while clauses C2, C3, and C5 may be stored in a Secret file. If an Unclassified user poses a query, only the clauses C1, C4, and C6 are examined. If a Secret user poses a query, then all of the clauses (i.e., both files) are read and the resolution procedure is implemented. As in the case of the MLS/RDBMS, performance will be an issue with this approach as the clauses will have to be decomposed and stored in single-level fragments.

This approach also has another major disadvantage. Since a deductive manager has reasoning capability, it may be able to deduce information classified at a level higher than the level at which it operates. If the deductive manager is untrusted, then it could release this higher level information to the user. In other words, the user would have acquired information to which he is not authorized. Since the deductive manager is used as a reasoning component of an MLS/DEDDBMS, then its ability to detect unauthorized inferences could become a security problem especially if the deductive manager runs as a single-level untrusted subject. This means that if the inference problem is not of any concern, then one could envisage the deductive manager running as an untrusted subject. However, if the inference problem is of concern, then this architecture is not a viable one.

In the second architecture, that is the trusted subject-based architecture, the MLS/LPS is trusted to filter out data that is above the user’s level. In this case a file at level L could contain clauses at level L or below. The operating system controls access to the file. The MLS/LPS controls access to the individual clauses. That is, the clauses which are classified above the user’s level are not included in the resolution procedure. A major disadvantage with this approach is that the MLS/LPS (or at least parts of it) must be trusted. However, trusting the MLS/LPS would imply that such an architecture could be utilized to handle the inference problem.

Since we are concerned with the inference problem, we assume that certain parts of the deductive manager are trusted. That is, the deductive manager is trusted to: (i) filter out higher level clauses in the resolution procedure, and (ii) implement the security policy which is enforced to handle certain types of inferences. Our prototype will be described in the next section.

4. A PROTOTYPE IMPLEMENTATION

As stated in the previous section, there are two tasks that the deductive manager must perform. They are: (i) processing the clauses for problem solving, and (ii) handling the inference problem. In the first task, the clauses are processed by implementing the resolution procedure described in section 3. If a user at level L poses a query, then the clauses at level L and below are examined and the resolution procedure will be implemented in order to produce the response. If any data has to be obtained from the database, then the MLS/RDBMS will be invoked. Since the function of the deductive manager is that of a problem solver, one could envisage the operating system providing all of the mandatory security. That is, the clauses at level L* are stored at level L*. If a user’s request is at level L, then the deductive manager operates at level L and examines files containing clauses at level L or below. As stated in section 3, there could be a performance problem as the resolution procedure must be implemented across clauses in different files. If the trusted-subject architecture is used for the deductive manager, then the clauses at different levels could be stored in a file at the high water mark and the trusted deductive manager could select only those clauses classified at or below the user’s level in order to implement the resolution procedure.

Figure 7. MLS/DEDDBMS Prototype

In the second task, the deductive manager performs a security critical function which is to control unauthorized inferences. In this case, the deductive manager must be a trusted process since it can deduce higher level information from the information obtained from the database. It functions in such a way that certain information is not released to the user if it
detects that a user can make unauthorized deductions from the responses obtained. The deductive manager that we have implemented focuses only on controlling certain unauthorized inferences that users could make. The architecture of the prototype that we have implemented is shown in figure 7. The MLS/RDBMS that we have used in the implementation is Sybase's Secure SQL Data Server [SYBA89]. It is based on the trusted subject architecture described in section 2. It supports 16 security levels and provides classification at the tuple level.12 The deductive manager implements query modification and response sanitization techniques. The query modification technique utilizes the resolution procedure described in section 3. The response processing technique is implemented only to control certain unauthorized inferences which depend on the user's prior knowledge (which is the information already released to the user). The multilevel knowledge base consists of the security constraints, integrity constraints, real-world information, and history information.

The user's query is posed in logic. The deductive manager modifies the query. The modified query is translated into the relational language SQL. The relational query is evaluated against the multilevel relational database. We illustrate the essential points with examples. Consider a database which consists of a relation EMP where the attributes of EMP are SS#, Ename, Salary and Dept# with SS# as the key. Let the knowledge base consist of the following clauses which we assume are Unclassified.

1. \( \text{Level}(Y, \text{Secret}) \rightarrow (\text{EMP}(X,Y,Z,D) \land Z > 70K) \)
2. \( \text{Level}(Y, \text{Top Secret}) \rightarrow (\text{EMP}(X,Y,Z,D) \land D = 30) \)
3. \( \text{Level}(X, \text{Unclassified}) \rightarrow (\text{NOT}((\text{Level}(X, \text{Secret}) \lor \text{Level}(X, \text{Top Secret})))) \)

The first clause classifies a name whose salary is more than 70K at the Secret level. Similarly, the second clause classifies a name whose department is 30 at the Top Secret level. The third clause states that if an item is not Secret or TopSecret, then it is Unclassified.

Suppose an Unclassified user queries for the names in EMP. This query is represented as follows: \( \text{EMP}(X,Y,Z,D) \) and \( \text{Level}(Y, \text{Unclassified}) \).

The proof procedure of the deductive manager uses the algorithm described in section 3. That is, the deductive manager will start with the query and perform appropriate substitutions for the various predicates occurring in the query. The following steps will be included in the query modification process:

Step 1: \( \text{EMP}(X,Y,Z,D) \) and \( \text{NOT}\left(\text{Level}(Y, \text{Secret}) \lor \text{Level}(Y, \text{Top Secret})\right) \);
Step 2: \( \text{EMP}(X,Y,Z,D) \) and \( \text{NOT}(Z > 70K) \lor (D = 30) \);
Step 3: \( \text{EMP}(X,Y,Z,D) \) and \( Z <= 70K \) and \( D <= 30 \).

The modified query is \( \text{EMP}(X,Y,Z,D) \) and \( Z <= 70K \) and \( D <= 30 \).

The prototype can also handle logical constraints such as those of the form "A IMPLIES B". If B is Secret and A is Unclassified, the the deductive manager will ensure that A is not released to an Unclassified user.13

In addition to query modification, the deductive manager also performs response sanitization. This function is not part of the resolution process and is only necessary if the system must be protected against inference threats. We illustrate the essential points with an example. Consider the following release constraint:

- all salaries whose corresponding names are already released to Unclassified users are Secret.

When an Unclassified user queries for names, the names in the response together with the corresponding SS# values will be recorded in the release database. Later, when the salaries are queried by an Unclassified user, the DBMS is queried for the salary and SS# values. The deductive manager then compares the response with the information in the release database. It only releases the salary if the corresponding name has not been released.

There are some problems associated with maintaining the release information. As more and more relevant release information gets inserted, the knowledge base could grow at a rapid rate. Therefore efficient

12 We used this MLS/RDBMS as it was available for our use at the time of the implementation.

13 We have only described the essential points of the prototype. The prototype can handle complex constraints. A more detailed discussion of the various types of constraints, the modules of the deductive manager, and the trust that must be placed on the various modules are given in [FORI89]. We have extended the prototype to a distributed environment. The enhancements are not discussed in this paper.
techniques for processing the knowledge base need to be developed. This would also have an impact on the performance of the query processing algorithms. Therefore, one solution would be to include only certain crucial release information in the knowledge base. The rest of the information can be stored with the audit data which can then be used by the Systems Security Officer for analysis.

5. CONCLUSION

In this paper we have described security issues for MLS/DEDBMSs. We defined an MLS/DEDBMS to be an MLS/RDBMS augmented with a deductive manager which is a logic programing system for a multilevel environment. We provided an overview of MLS/RDBMSs and then described in detail the multilevel logic programming system which acts as the deductive manager. In particular, a logic model for multilevel databases as well as reasoning in a multilevel environment were described. The essential point here is reasoning across security levels. We discussed architectural issues for an MLS/DEDBMS and finally described a prototype implementation.

The work reported in this paper is just the first step towards developing an MLS/DEDBMS. Much work remains to be done before fully functional MLS/DEDBMS can be developed. First of all, our prototype is tailored only to processes security constraints in such a way that certain security violations via inference do not occur. That is, our prototype does not handle arbitrary clauses. The logic that we have developed is also limited because it simply assumes that information at the higher level is always true. In many cases this may not be the case as higher level information may be outdated. In such cases, the resolution procedure should be able to reason in such a way that it takes into consideration the most accurate information. In other words, our logic must be reexamined and subsequently extended so that it can model real world situations more accurately. Subsequently, a logic programming system based on the logic needs to be developed.

REFERENCES


