Secure Software Architecture Design for Multidatabse System

By

Harshavardhan Reddy Mogatala

A Thesis

Submitted to the Faculty of Graduate Studies at University of Manitoba

in Partial fulfillment of the Requirements

for the Degree of

MASTER OF SCIENCE

Department of Computer Science

University of Manitoba

Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

Secure Software Architecture Design for Multidatabase System

BY

Harshavardhan Reddy Mogatala

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of

Master of Science

Abstract

This thesis discusses security issues related to multidatabase system and proposes a prototype secure software architecture design. The motivations behind developing the architecture design are to provide multilevel security, resolve the cascade problem, and develop the security architecture according to trusted computer standard evaluation criteria by US Department of Defense (DoD). The main goals in the design are illustrated by means of a multilevel security model, data protection concepts, and solving the cascade problem. The major contribution of this thesis is the development of mechanisms that provides a transparent security architecture for multidatabase systems.

Acknowledgments

There are many people that I would like to thank. First I would like to thank my co-supervisors Dr. Ken Barker and Dr. Sylvanus Ehikioya for guiding me through out my thesis and being there when I needed them. I would also like to thank the members of my thesis committee Dr. Ken Barker, Dr. Sylvanus, and Dr. Jose Rueda. And Dr. John van Rees for chairing my thesis defense.

Another "Thank You" to Dr. Jose Rueda of TR*Labs* for providing me with a great research environment. Thank you also for providing me moral and technical support through out my research period at TR*Labs*. Other people that were helpful in the completion and correction of my thesis are Phi Thang, Grace Liu and others TR*Labs* members.

Also to be thanked in general are my parents and sister.

Finally, a thank you to TR*Labs* as an organization for providing the necessary financial support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There have been several important paradigms for distributed computing in recent years. Progress in communication and database technologies have drastically changed user data processing capabilities. The need for the integration of data from heterogeneous and physically distributed information sources has triggered research and development in the area of multidatabase systems (MDBS). A multidatabase [2] or federated database system [37] allows users to access data located in multiple autonomous and possibly heterogeneous local databases (LDBS). Local transactions (confined to a single database) are submitted directly to the LDBS, while the global transactions (not confined to a single database) are channeled through the MDBS interface.

1.1 Motivations

Although the increasing widespread use of both centralized and distributed databases have proven necessary to support business functions, they also pose serious problems for data security. In fact, a security breach in a multidatabase environment not only affects a single user or application but also the whole information system [31]. Therefore, in multidatabase information systems, secure architecture design is essential to assure both system availability and reliability. Research on secure architecture design has been an ongoing process for several decades [34]. The key features of this past work are typified by their important pieces of work.

Thuraisingham and Ford [41] developed an approach for security constraint processing (security constraints are used to assign security levels to the data based on content, context, and time) in a multilevel secure distributed database management system. They used security constraints as an effective classification policy and handle the constraints during query processing, database updates, as well as database design. They also described an integrated architecture that shows the interactions between the query constraint processor, the update constraint processor, and the database design tool.

Qian and Lunt [43] developed a semantic framework of the multilevel secure relational model with tuple-level labeling, which formalizes the notion of validity in multilevel relational databases. They also provided the semantics for the multilevel secure relational model that preserves both integrity and secrecy.

Sandhu [40] provided a basic architectures for multilevel security database management systems, and showed how the nature of a distributed multilevel security database system is influenced by the nature of the network and distributed system architecture on top of which it is built.

1.2 Objective of the Study

The main objective of the thesis is to provide additional security to a MDBS. To provide security in multidatabase systems, researchers have relied on a security model

called multilevel security (MLS) [40]. In a multilevel secure database management system, users cleared at different security levels, access and share a database consisting of data at different sensitivity levels. An approach to assigning sensitivity levels (also called security levels) to data is one that utilizes security constraints or classification rules. Security constraints provide an effective classification policy. They can be used to assign security levels to the data based on content, context, and time. Such constraints are useful for describing multilevel applications.

The complexity of the design and the implementation of a secure multidatabase system depends on several factors, such as the heterogeneity of systems, the distributed environment, multilevel security, and the difficulties in modeling, specifying and verifying data security. Secure database management systems (DBMS) operate in one of two possible modes [34]: *the high mode* or *the multilevel mode*. In high mode, all the users are cleared to the highest security level. A system administrator (human guard) is responsible for reviewing the user's query before the user is granted permission to access the database. This approach allows users to use existing DBMS technologies without changes, but generates additional costs because of the clearance procedure and the need to manually review data [4]. This approach is mostly used in secure database systems like the Top Secret Worldwide Military Command and Control System (WWMCCS) [15], which operates in high mode.

The multilevel mode allows information with different sensitivities (i.e., classification) to be simultaneously stored and processed in an information system

with users having different security clearances and authorizations. It permits different type of architectures, based on the use of both trusted and untrusted DBMSs [10]. Multilevel architectures are suitable for different purposes based on characteristics and the requirements of the target application domain. For example, the kernelized architecture [1] fits environments requiring single-level tables because it is easily implemented and very economical. For environments characterized by a DBMS that requires "label flexibility" (i.e., the security label can be physically or logically attached to the data.) and a high degree of integration between the DBMS and the underlying operating system (OS), the integrity lock architecture should be utilized. The Trusted Subject Architecture [10] is suitable for application domains where a trusted path can be assured from applications to the DBMS.

For the purpose of this thesis, the kernelized architecture is used as the reference model. The kernelized architecture is ideal because it satisfies the three important components of security, namely, secrecy (by implementing the Bell-LaPadula model [10]), integrity (secure concurrency control protocol must guarantee serializability [10]) and availability (secure concurrency control protocol should not cause starvation [10]). These three important aspects are also part of the U.S Department of Defense evaluation criteria for a secure system [10].

1.3 Contribution

This thesis proposes a new secure software architecture (SSA) for the multidatabase environment. This secure software architecture addresses problems that arise due to

the distributed environment, such as the need for authorization controls and multilevel security.

The main contributions of this thesis are:

- Provides a simple, secure, and transparent architecture that can be implemented on top of existing multidatabase systems
- Presents a new approach to provide multilevel security on top of a relational multidatabase system.
- Provide solution to the cascade problem, which exists when a user can take advantage of network connections to compromise information across a range of security levels that is greater than the user's security level [23].
- The architecture design is simulated and tested with the help of security components providing the interface to the multidatabase system.

The security architecture model proposed in this thesis uses the criteria defined by the US Department of Defense [15] as a reference for both the development and evaluation of the new secure software architecture. These criteria will be discussed in later chapters and will be used as a benchmark to evaluate our architecture.

The architecture proposed in this thesis consists of four major security components, which are interfaced with the multidatabase system to provide multilevel security and solve the cascade problem. The four security components are Global Security Administrator, Local Security Administrator, Global Data Catalog, and Local Data Catalog. Their details are discussed in Chapter 3. The architectural design is

simulated using a homogeneous relational data model with the four major security components providing the interface to the multidatabase, data filtration, and user-data level security functions. Different SQL query transactions were performed on the relational database system with varying user and data security levels to test the system. In addition, the system administrator is provided with system tools to create user profiles in the database that store information about the user security level, access privileges and security policy for the database.

1.4 Limitation

The SSA does pose some limitations. While developing the SSA, more emphasis is put on the security of the database and the cascade problem so efficiency and accessibility of the system are sacrificed. The relational database system (InstantDB) used in developing the SSA has its own limitations. Some of the functions that are missing in the system, such as outer joins, database views, multiple user names or privileges, GROUP BY and HAVING. Further, there are no boolean data types. Although efficiency and performance are very critical to real-world application the focus here is in understanding the core issues of MDB security. The performance issues can be addressed once a thorough understanding of how to build a secure MDB environments is established.

1.5 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 provides an overview of related research and different multidatabase system architectures. The new secure

software architecture design for a multidatabase system is proposed in Chapter 3 and the implementation details are provided with examples in Chapter 4. Chapter 5 contains concluding remarks and suggestions for future directions on the secure software architecture both at the implementation and design stage.

Chapter 2

Related Research

This chapter gives a brief introduction to multidatabase system, including the definition, characteristics, and security features that relate to the current research. It offers a comparison of different MDBS architectures and their security approaches.

2.1 Multidatabase System Architectures

A multidatabase system is a facility that supports global applications accessing data stored in multiple databases. It is assumed that access to these databases is controlled by autonomous and possibly heterogeneous Local Database Systems (LDBSs). The MDBS architecture (Figure 2.1) allows local and global transactions to coexist. Local transactions are submitted directly to a single LDBS whereas the multidatabase (global) transactions are channeled through the MDBS interface [21].

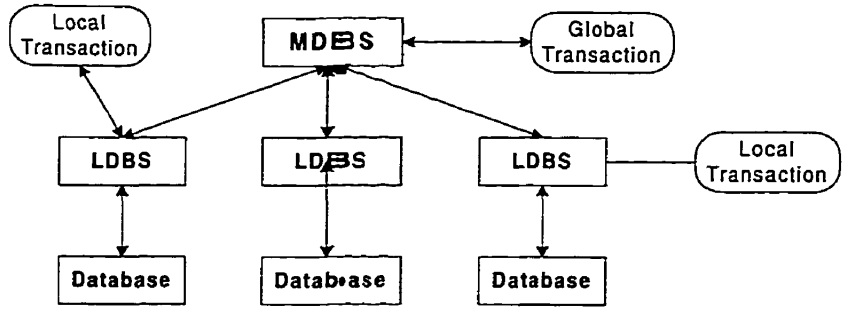

Figure 2.1 Multidatabase System

A multidatabase system acts as a front-end to multiple local DBMSs or is structured as a global system layer on top of local DBMSs. The global system gives full database functionality and interacts with local DBMSs at their external user interface.

Although each local node must maintain some global functions to interact with the global system, the local DBMSs are autonomous [21]. The global system provides some means (global schema or multidatabase language) of resolving differences in data representation and functionality between local DBMSs. This resolution capability is necessary because the same information may be maintained at multiple locations in different forms. Thus, users can access information from multiple sources with a single query.

Characteristics

The following aspects characterize a multidatabase system [32,2].

- *Autonomy:* An essential characteristic of the multidatabase is that the local DBMSs remain autonomous; that is they retain full control over local data and processing. Each local DBMS independently determines what information it will share with the global system, what global requests it will service, when it will join the multidatabase, and when it will stop participating in it. Local autonomy is a virtue of the multidatabase because it allows global access to be added to existing DBMSs with minimal impact on local processing.

- *Heterogeneity:* Data models, query languages, and processing methods may differ among multiple DBMSs. An enterprise may have multiple DBMSs. Different organizations within the enterprise may have different requirements and may select different DBMSs. Database management systems purchased over a period of time may be different due to changes in technology. Heterogeneity due to differences in DBMSs result from differences in data models. In addition, heterogeneity due to differences in data representations can result from name differences, format differences, structure differences, abstraction differences, and missing or conflicting data.

- *Distribution:* Data may be distributed among multiple databases. These databases may be stored on a single computer system or on multiple computer systems, co-located or geographically distributed but interconnected by a communication system. Multiple copies of some or all of the data may be maintained. However, distribution may not always be a necessary condition to justify an application of a MDBS.

- *Logical correlation:* Logical correlation implies the existence of a common logical data model with its corresponding data language. Since many different data models already exist, corresponding to the various centralized databases accessible in the realm of a multidatabase system, the necessary mappings between those data models and the common data model of a multidatabase system must be provided.

- *Global concurrency control:* Concurrency control is the coordination of concurrent processes that operate on shared data and may potentially interact with each other. Centralized DBMSs typically use a lock on a data item to indicate one process is updating the value and other processes must wait until the update completes before the new value can be applied. Multidatabase transactions will typically involve multiple, separate local DBMSs and several layers of data/query transactions. More importantly, local DBMSs have site autonomy so global control does not include control of the actual data items.

- *Global query processing:* The basics of global query processing are consistent across most multidatabases. A user employs the global schema to submit a global query, or in the case of a multidatabase language, the query itself contains all the information necessary for retrieving local data. The query is decomposed into a set of subqueries, one for each local DBMS that will be involved in the query execution.

The query optimizer creates an access strategy that specifies which local DBMSs are to be involved, what each will do, how to combine the intermediate results, and where global processing will occur. Finally, in global execution process queries may be translated several times as they travel through the various system layers before being executed.

2.2 Different Multidatabase System Architectures

Several MDBS architectures have been proposed in the literature. In the following discussion these architectures are considered. There are two types of multidatabase models reported in the literatures: global conceptual schema architectures and non-global conceptual schema architectures.

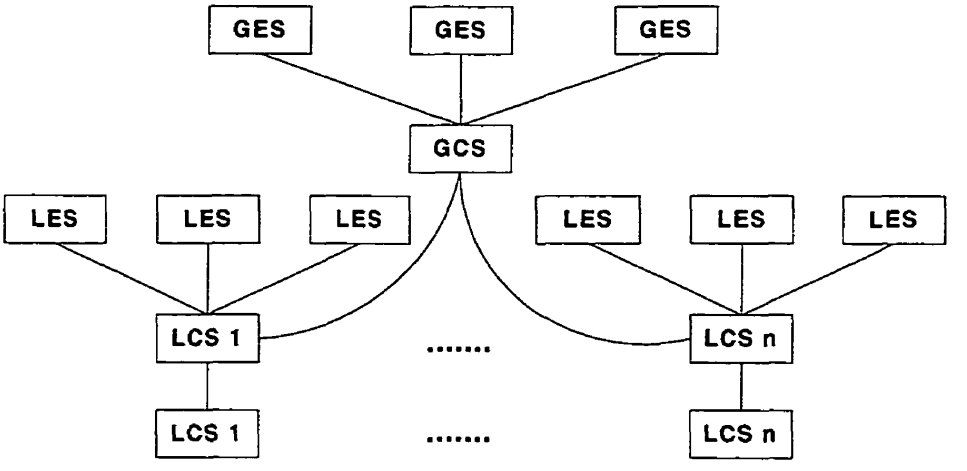

Figure 2.2 MDBS Architecture with GCS

2.2.1 The Global Conceptual Schema Architectures

Architectures using global conceptual schema (GCS) integrates either the local conceptual schemas or a set of local external schemas. The GCS is constructed using a bottom up technique and if heterogeneity exists in the MDBS system, a canonical data model must be defined for the GCS [32] (see Figure 2.2).

After the GCS has been designed, global users can create global views. Once again the global external schema and GCS need not use the same data model and language. There are different architectures available in the literature based on the above model. We will discuss a representative sample below.

- **IMDAS**

The Integrated Manufacturing Database Administration System (IMDAS) has been developed by the National Institute of Standards and Technology [42]. IMDAS supports only relational databases meant for manufacturing systems. The computing environment consists of a network of heterogeneous component systems acquired from various vendors. The system design envisages that the integrated databases contain all data needed for user access. In addition, this database should serve as a medium through which the component systems communicate with one another. IMDAS provides local autonomy, which is essential in a manufacturing environment because the control and data systems must provide for the independent testing and modular integration of new equipment and control software. The schema architecture of IMDAS contains four layers as show in Figure 2.3.

At the bottom level of the IMDAS are the data repositories such as commercial DBMSs. These DBMSs are logically clustered into groups. The interface between DBMSs of each group and the rest of the IMDAS is provided by the Basic Data Administration System (BDAS). With the use of a dictionary describing the distribution of data at subordinate BDAS, the Distributive Data Administration System (DDAS) takes responsibility for all such hosts and their corresponding data. To resolve conflicts between various DDAS, a single system is designated as the Master DAS (MDAS). To provide a uniform interface to the integrated database a

common global data model called the Semantic Association Model (SAM*) is used. Database requests are made using a common global data manipulation language (DML). When data incompatibilities and semantic mismatches arise, IMDAS performs conversions and maintains relationships between dependent relations and parent relations by using the Global Schema. IMDAS has an online SAM*

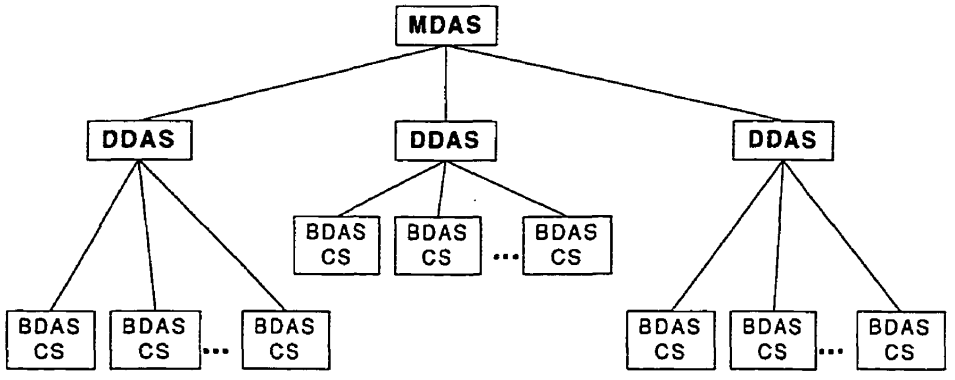

Figure 2.3 IMDAS Architecture

Dictionary utilities for defining Global Schema, Views, Site capabilities, Fragmentation, and Partitioning.

A query on the Global External View is modified and converted to queries on the Global Schema. If Global Schema manager (DDAS) cannot process the query, it is passed to the Master Schema Manager (MDAS) for further processing; otherwise, it is translated into queries on local schemas.

- **PRECI**

Prototype of a Relational Canonical Interface (PRECI) [14] is a prototype MDBS developed at the University of Keele (UK). Local data models supported by PRECI support any local model via a relational algebra interface. The schema architecture of PRECI is as shown in Figure 2.4. One peculiar aspect of PRECI is its support for two

varieties of nodes: inner nodes and outer nodes. Inner nodes provide the best available service to global users through global database schema and global external schema, and also facilities like location transparency and replication transparency. These facilities are not available for data at outer nodes because only data from inner nodes can participate in the global schema. The lowest layer NES represents the nodal external schema that can be used by the local user. Each nodal DBMS must support at least a minimal subset of PRECI Algebraic Language (PAL), viz., selection, projection, join, division, union, and difference. The Participation Schema (PS) describes the nodal data along with various authorization controls. Each participation schema has a version number, which is updated each time the PS is changed.

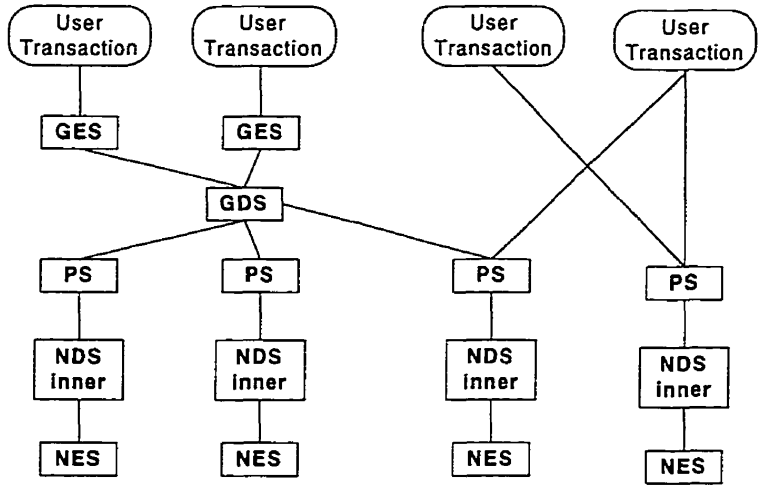

Figure 2.4 Schema Architecture for PRECI

The knowledge required to resolve data inconsistencies and the details of replication of data are stored in a separate database called the Subsidiary Database (SDB), which is managed by the SDBMS (Subsidiary Database Management System) under the control of the DDBMS at each node.

The Global Database Schema (GDS) is formed by the PS of the inner nodes. The GDS supports the integration of data and metadata stored in the SDB. The GDS itself

does not provide an integrated view. Instead, all conversion formulae and other relevant information are stored. The desired integration is carried out at the Global External Schema (GES).

2.2.2 Architecture without a Global Conceptual Schema

Let us consider MDBS architectures without a GCS. In the MDBS architecture shown in Figure 2.5, there are only two layers: the local system layer and the multidatabase system layer.

The local system layer consists of a number of DBMSs, which present to the multidatabase layer the part of their local database they are willing to share with users of other databases. This shared data is presented either as the actual local conceptual schema or as a local external schema definition. Figure 2.5 shows these as a collection of local conceptual schemas.

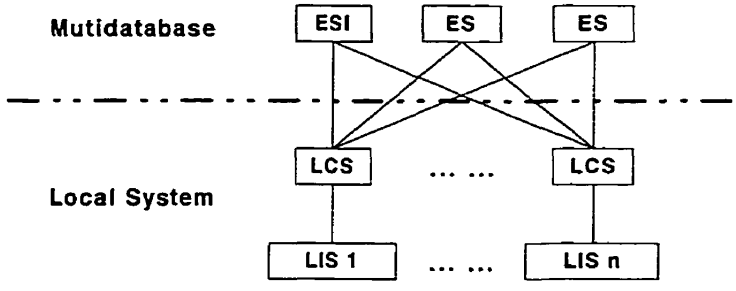

Figure 2.5 MDBS Architecture without a GCS

In case of heterogeneity, each of these schemas may use a different data model. Above this layer, external views are constructed where each view may be defined on one local conceptual schema or on multiple conceptual schemas. Thus the responsibility of providing accesses to multiple (and possibly heterogeneous) databases is delegated to the mapping between the external schemas and the local conceptual schemas. In practice, access to multiple databases is provided by means of

a powerful language in which user applications are written. Let us consider some systems that do not use a GCS.

- **MRDSM**

Multics Relational Data Store Multibase (MRDSM) [30] was developed by INRIA (France) to support multiple databases designed using the Multics Relational Data Store (MRDS) relational database management system of Honeywell.

Heterogeneity is dealt with at the semantic level by providing uniform access to all databases implemented with the same DBMS. The global query language, MDSL, is an extended version of DSL, which is the data manipulation language of MRDS. The schema architecture of MRDSM is shown in Figure 2.6. Note that the global schema does not exist in MRDSM. Users can create conceptual schema known as multischema with elements from local database schemas along with one or more dependency schemas to handle interdatabase dependencies. The elements of multischema created by the user are: (i) one or more dependency schema, or (ii) explicit enumeration of local databases to compose a multidatabase.

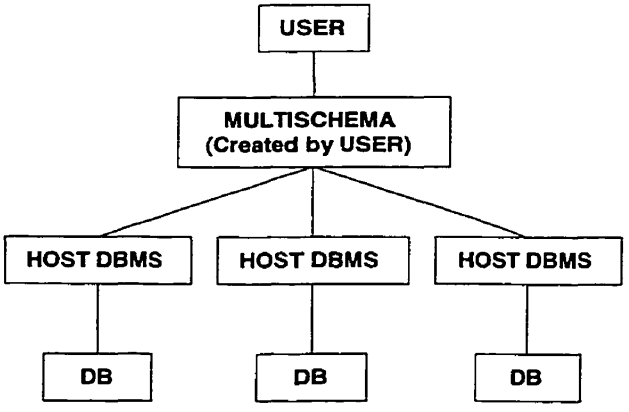

Figure 2.6 Schema Architecture for MRDSM

A query on multischema is decomposed into queries on local databases after removing

interdata dependencies that cannot be handled locally. Subsequently, a working DB is created to collect data from different DBs. The collection process is optimized by performing projection and selection operations on source DBs. Finally, queries are generated on the working DB to combine data.

- **DB2 DataJoiner**

DB2 DataJoiner is IBM's strategic offering to enable transparent, consistent, single-command access to heterogeneous multi-vendor data sources [12].

It enables users to join data from disparate databases through a single SQL statement and a single interface, hiding the various database differences from the user and the user's application. The user does not even have to know where the data resides. DB2 DataJoiner provides point-to-many points access to several remote data sources at the same time.

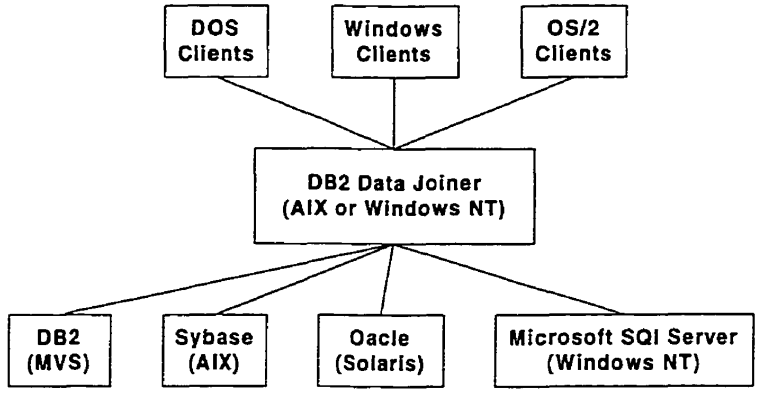

Figure 2.7 DB2 DataJointer and Its Supported Clients and Data

Figure 2.7 illustrates a multidatabase server configuration where DB2 DataJoiner can access data stored in Oracle, DB2, Microsoft SQL Server, and Sybase files. In addition to the DOS, Windows and OS/2 clients shown, DB2 DataJoiner also supports Macintosh and various Unix clients.

A key to DB2 DataJoiner's efficiency is its use of global optimization technology for efficiently performing heterogeneous queries. DB2 DataJoiner masks the differences in various vendor dialects of SQL and manages the complexity of establishing connections to different data sources, translating requests into the native interfaces of these data sources, coping with differences in data types, and determining an efficient data access strategy to satisfy the request. DB2 DataJoiner also provides for sophisticated operations such as transparent, distributed joins and unions of data from multiple sources.

2.3 Security in Multidatabase Systems

A multidatabase system integrates distributed, (possibly) heterogeneous databases while preserving their autonomy. The integration is often a compromise between the users explicitly interfacing with several databases and their desire to access distributed data through a single, global interface. A MDBS provides users with the capability to retrieve data located at different sites and/or companies and to integrate this data within a global view as transparently as possible.

Although there is significant advantage to interoperability, the need to protect the privacy of local databases and their local users increases dramatically. Most of the mechanisms ensuring privacy and security requirements add new functionality to the local systems. Some of the important security issues for a MDBS are described below.

- *Identification and Authentication*

The MDMS may wish to authenticate the local sites to which a connection for global users is requested. In turn, each local site may wish to authenticate the MDBS site to which it should offer its local data.

- *Authorization and Access Controls*

Data granularity (e.g. users, roles, etc.), access types, and database objects may be heterogeneous or homogeneous among the LDBSs of a mutidatabase system. A MDBS has to provide the level of granularity that allows suitable identification at various local systems. A powerful security model must provide the MDBS with a global security policy that can tolerate local security policies.

- *Integrity and Consistency Constraints*

The data model of the MDBS has to provide the means to specify multi-site integrity and consistency constraints because the data offered by the multidatabase may be scattered over various local sites.

- *Audit*

Security auditing, like identification and authentication, has to be performed at the multidatabase site as well as at the various local sites. Additional audit records have to be generated if control passes from the global layer of the multidatabase to a particular LDBS at the local layer.

- *Development Reliability*

The MDBS is a distributed system, so testing the system security will require techniques that are tolerant to a typical network environment such as site loses or network partitioning.

• *Data Encryption*

The MDBS may use multiple encryption techniques so the system must ultimately be designed to handle any number of different encryption protocols. Clearly this will require a combination of industrial standards and peer-to-peer communication protocols such as International Standards Organization / Open Systems Interconnection model (ISO-OSI).

2.4 Cascade Problem

The U.S. Department of Defense's evaluation criteria for a secure system includes three main categories: s*ecurity policy, accountability, and assurance* that are described below. In case of *security policy* both discretionary access control (DAC) and mandatory access control (MAC) in addition to a label mechanism [11] are required. With DAC, subjects (users or groups of users) can protect their own objects and can grant or deny access permissions to other subjects. In DAC, subjects can read/write objects for which they have the required clearance. MAC operates using security labels that are assigned to both subjects (security clearances) and objects (security classification). Security labels specify a hierarchical classification and a set of non-hierarchical categories that together state the clearance of the subject and the level of sensitivity of the information stored in the object. According to MAC, subjects can read objects at the same or lower classification, but can write objects at the same or higher classification. Unfortunately, MAC raises a security issue called the *cascade problem* [23]. The cascade vulnerability problem occurs when independent mutually recognized secure systems are interconnected by a secure channel to create a distributed system. Although both systems are secure the connection is not assumed to be safe. "A cascading problem exists when a penetrator can take advantage of network connections to compromise information across a range

of security levels that is greater than the accreditation range of any of the components systems s/he must defeat to do so" [23].

As a typical example of the cascade problem, let us consider two systems, as shown in Figure 2.8. Host A is accredited for TS-Top secret and S-Secret level information and all users are cleared to at least the Secret-level. Host B is accredited for Secret and Confidential-level and all users are cleared to at least the Confidential-level; finally, there exists a link at the Secret level between the two systems. While the risk of compromise in each of these systems is small enough to justify their use with two levels of information, the system as a whole has three levels of information. This increases the potential harm that an adversary could cause, because the adversary could downgrade Top-Secret information in Host A to Secret-level, send it to Host B through a secure channel and further downgraded the information to Confidential-level. The fact that the adversary downgrades the security level is the real problem, but in case of multidatabase systems the network connection between the databases increases the potential harm she/he can cause.

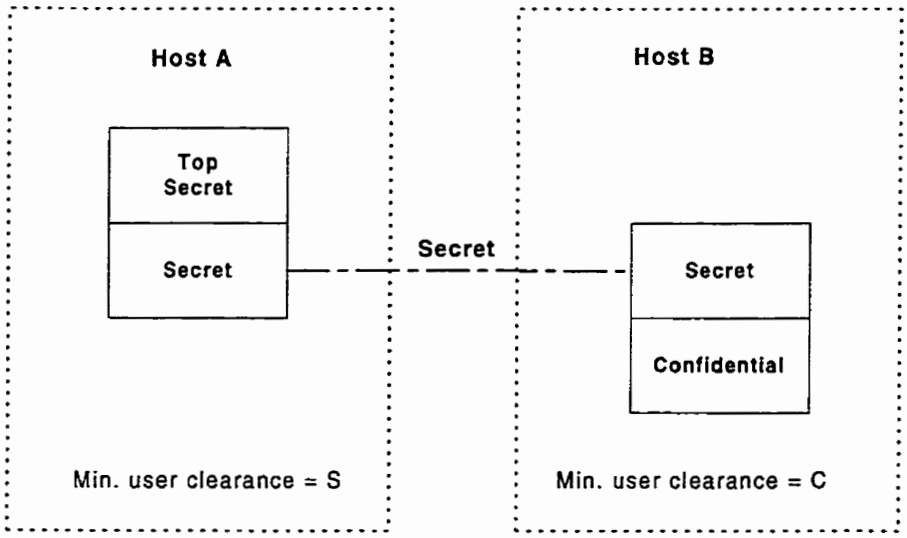

Figure 2.8 The Cascading Problem [6]

In a multidatabase system with a multilevel security model, the adversary has to defeat the protection mechanisms of both Host A and Host B because he has to defeat only two security levels Secret and Top-Secret or Secret and Confidential (only 2 levels). But that is an easier job than defeating the protection mechanisms of a single system trusted to protect the whole range from Top-Secret to Confidential level (4 or more security levels). The network connection has, in essence, created a Trusted Computing Base (TCB) [11] with users cleared to at least the Confidential level with data on it at the Top-Secret level. The trusted computing base is the part of the system that is responsible for enforcing the information security policies of the system. All of the computer's hardware is included in the TCB but the system administration should be concerned primarily with the software components of the TCB [11]. In this way the network connection has invalidated the risk analysis that accredited the two systems. In addition to this, security risk is further enhanced by the large amount of data that is available for the adversary to downgrade because multidatabase system consists of many databases interconnected to one another through network so the potential damage is more. Because of the above two factors such a system must have a more secure architecture. Hence, multidatabase systems are vulnerable to the cascade problem.

The *accountability* issue includes the features of identification/authentication, audit, and trusted path.

In the *assurance* issues, system must contain reliable hardware/software/firmware components that can be evaluated separately to ensure the system can meet the requirements of the previously listed categories. The assurance requirement can be partitioned into two further categories related to operation reliability and development reliability. Operation reliability deals with reliability during system operation, with

the purpose of assuring trusted execution of security features and development reliability concerns with system reliability during the development process.

2.5 Security Architectures and Models Review

In this section, we will review five different security architectures and models that are considered while designing the secure software architecture. These security architectures and models describe in this section provides necessary groundwork for developing a secure system.

2.5.1 Security Architecture of IRO-DB

Gardarin, *et al.* [22] describe a security architecture of the IRO-DB (Interoperable Relational and Object-Oriented Databases) database federation, which is a system supporting interoperable access between relational and object-oriented databases. IRO-DB uses a three layered architecture having a local, communication, and an interoperable layer. A common object-oriented data model is used and has been extended to provide interoperability among relational and object-oriented databases. A security model is developed that can be characterized as federated, administrative, discretionary access controls (FADAC) with the following attributes:

- **Discretionary** means that the creator of the object owns each database object. Owners have unrestricted access to their objects but, contrary to other discretionary policies, have no right to delegate access to others.

- **Administrative** determines that the delegation of authorizations is under the control of administrative authorities. The authors used role-based access controls requiring users to play different roles to access different objects. Some of the roles have administrative rights for granting authorizations.

• **Federated** requires a mapping between local and global security concepts. Each local database makes its local security concepts appear to be compliant to FADAC. Powerful access control mechanisms are provided including positive, negative, and implied authorizations with the aim to ensure consistency between global and local authorization states. This approach allows the definition of global subjects (users and roles) that represent a mapping between a number of local subjects, one for each component database (CDBS). A global user may then choose among a set of global roles so requests can be issued against the federated database. Each request must satisfy the global security policy (checking for the global subject at the interoperable layer) and must not violate any of the local security policies by checking for the corresponding local subjects at the local layer where each of the engaged CDBSs are accessed.

This architecture ensures the privacy needs of the local system because it permits specification of the roles and the users visibility to the federation. Furthermore, it guarantees the security policies at each site participating in the federation and allows additional restrictions at the federation sites to govern unauthorized aggregation and inference from the local data.

2.5.2 The Jajodia and Sandu Model

Jajodia and Sandhu [25] proposed a model for the application of mandatory policies in relational database systems. The model extends the standard relational model of security classifications and provides a formalization of such an extended model. The model is based on the security classifications introduced in the Bell-LaPadula model [11]. They considered two requirements: entity integrity and update semantics.

In classical relational theory the essential constraints have been identified as entity integrity and referential integrity. Jajodia and Sandhu [25] generalizes the usual entity integrity requirement to a multilevel context. They also discussed the importance of "relation updates" to achieve secrecy of information in multilevel systems.

2.5.3 Secure System Software Architecture

Mark *et al.* [31] proposed a new approach to secure system design for an open architecture standard. The approach was illustrated by means of the X/Open distributed transaction processing (DTP) reference architecture. The X/Open DTP reference architecture allows multiple application programs to share heterogeneous resources provided by multiple resource managers and allows their work to be coordinated into global transactions. A formal approach to secure architectures is proposed that involve three steps:

- Formalization of the system architecture in terms of common architectural abstractions.
- Refinement of the system architecture into specialized architectures, each suitable for implementation under different assumptions about the security of the system components.
- Provides proof that every implementation that conforms to the system architecture, or one of its specialization, satisfies the intended security policy

The new approach was illustrated by presenting excerpts from their formalization of a secure version of the X/Open DTP standard called SDTP. It consists mostly of structural information involving component interfaces and the connections among them. An important contribution of this work was modeling of a system in terms of

multiple secure architectures that are related by formal mappings.

2.5.4 Transaction Processing in Multilevel Secure Databases with Kernelized Architecture

Atluri, Jajodia and Bertino [1] identified the challenges posed by multilevel security on transaction processing. They discussed how the three components of security, namely, secrecy, integrity, and availability, impose restrictions on database concurrency control.

- Secrecy Requirement -- A secure concurrency control protocol must be free of signaling channels, a type of illegal information flow channels that can be used by malicious transaction to leak sensitive information.
- Integrity Requirement -- A secure concurrency control protocol must guarantee serializability. This means that the effect of concurrent execution of transactions is equivalent to that of the transaction executed serially.
- Availability Requirement -- A secure concurrency control protocol must not cause starvation.

They also provided solutions to secure readers/writers problem that employ optimistic concurrency control techniques, "Secure Lock-Based Protocols" and "Secure Timestamp-Based Protocols". The advantages and disadvantages of the techniques with respect to transaction processing in multilevel secure databases is also outlined.

2.5.5 Encryption Model

The Encryption Model was proposed by the U.S Department of Defense U.S under the multilevel security program to promote the development and implementation of multilevel security solutions for information systems [15]. The goal of the program is

to develop, acquire, and deploy solutions and technologies that will allow the department of defense to meet operational requirements for multilevel security in its automated information systems.

The Encryption Model uses different encryption schemes to encrypt each record in a table and a different key for each field. This features helps in preventing users from accessing data that is not accessible to them and if a user who accidentally receives sensitive data cannot interpret the data.

2.6 Comparison of Different Security Approaches

The foregoing discussion of secure architecture design shows several different approaches: kernelized architecture (partitioning architecture), replicated architecture, and the encryption model.

The kernelized architecture partitions a multilevel database into single-level databases whereby data at each security level is stored separately. This architecture is also called a *partitioning architecture* [1]. It uses a separate DBMS for each security level to manage data at or below that level. The trusted front end ensures that the users' queries are submitted to the DBMS with the same security level as that of the user, while the trusted back end ensures that a DBMS at a specific security level accesses data without violating the mandatory security policy. The trusted front end is an interface through which all the user security is verified. The trusted back end is an interface through which all the data security is verified.

The kernelized architecture imposes some additional inefficiency on some basic advantages of databases, such as elimination of redundancy and improving data accuracy. It also does not address the problem of high-level user who needs to access

some low-level secure data that must be combined with high-level secure data. Processing of a user's query accessing data from multiple security levels involves expensive joins that may degrade performance because the different levels of data are stored separately. On the other hand, since the kernelized architecture has separate DBMSs for each security level, concurrency control schedulers can be separate for each level. Therefore, the scheduler can be implemented with untrusted code and need not be part of the trusted computer base (TCB).

The replicated architecture stores data from each security level in separate containers. In addition to the data at that level, copies of data at all lower levels are also maintained at each level. This architecture also uses a separate DBMS to manage data at each security level. Thus, if a high level transaction wishes to read data from a low level, it will be given the replica of the low-level data maintained by this high level container. As a result, this architecture is impractical for a large number of security levels. On the other hand, the replication overcomes the object-materialization performance problem in other architectures. Data that is apparently (to the user) multilevel is physically stored together as a single-level object in a single-level database system and does not need to be materialized [29].

The critical difficulty with the replicated approach is correct replication. Conventional approaches to replication and the related concurrency control issues are well understood. Unfortunately, the conventional solutions are either unworkable in the context of multilevel security or at best introduce covert channels [1]. If data cannot be correctly replicated without introducing security flaws, then the replicated approach is not viable.

The encryption model ensures that if sensitive data is encrypted, a user who accidentally receives sensitive data cannot interpret the data. In this model, each

security level has its own encryption scheme and user's need proper security clearance to decrypt the data. The disadvantage of this model is that it is not foolproof because the user can mount a plaintext attack or substitutes the encrypted form of user's own data [11]. Introduces operational overhead and increases the key management problem. The plaintext attack disadvantage can be overcome by using different encryption for each record and a different key for each field. The cryptographically linking fields of a record can be accomplished by using a block chaining method (CBC, CFB, *etc.*) [15]. This scheme is effective when the security of the data is considered more valuable than the efficiency of the system and is generally used in high security area, such as the military or a corporate organizations.

Chapter 3

Prototype Secure Software Architecture Design

The Secure Software Architecture (SSA) described here models a relational MDBS architecture. A relational MDBS architecture integrates local relational databases into one single virtual relational database by providing a global schema and a single database manipulation language. The SSA is an improvement over the current existing MDBS architectures available in the literature in many ways, as we will see below.

SSA implements multilevel security and prevents the problem of cascade vulnerability that exists in a large number of multilevel secure database environments [23]. Recall, the cascade problem exists when a penetrator takes advantage of a network connection to compromise information across a range of security levels [23]. There are not many secure MDBS architectures available that utilize the concept of Multilevel-Security (MLS) [15]. The SSA is a prototype that incorporates multilevel security using a new type of MLS model in which security is provided at the attribute level. Note that the attribute level means each element in the table is provided security level at logical level[1] not at the physical level in the table.

Multilevel-security is achieved by developing different secure software components that provides a logical link between the physical database tuple and the security level.

[1] Logical level means there is a logical link between the data that are physically stored in the database and the security levels of the data physically stored in the data catalog.

This is desirable because it provides secure database access, solves the cascade problem, and provides transparency both at the user and data level.

3.1 Prototype Secure Architecture Design for MDBS

Figure 3.1 illustrates the SSA system and highlights the additional security components that distinguish this architecture from the multidatabase system architecture. Data independence is supported since the model is an extension of ANSI/SPARC [32].

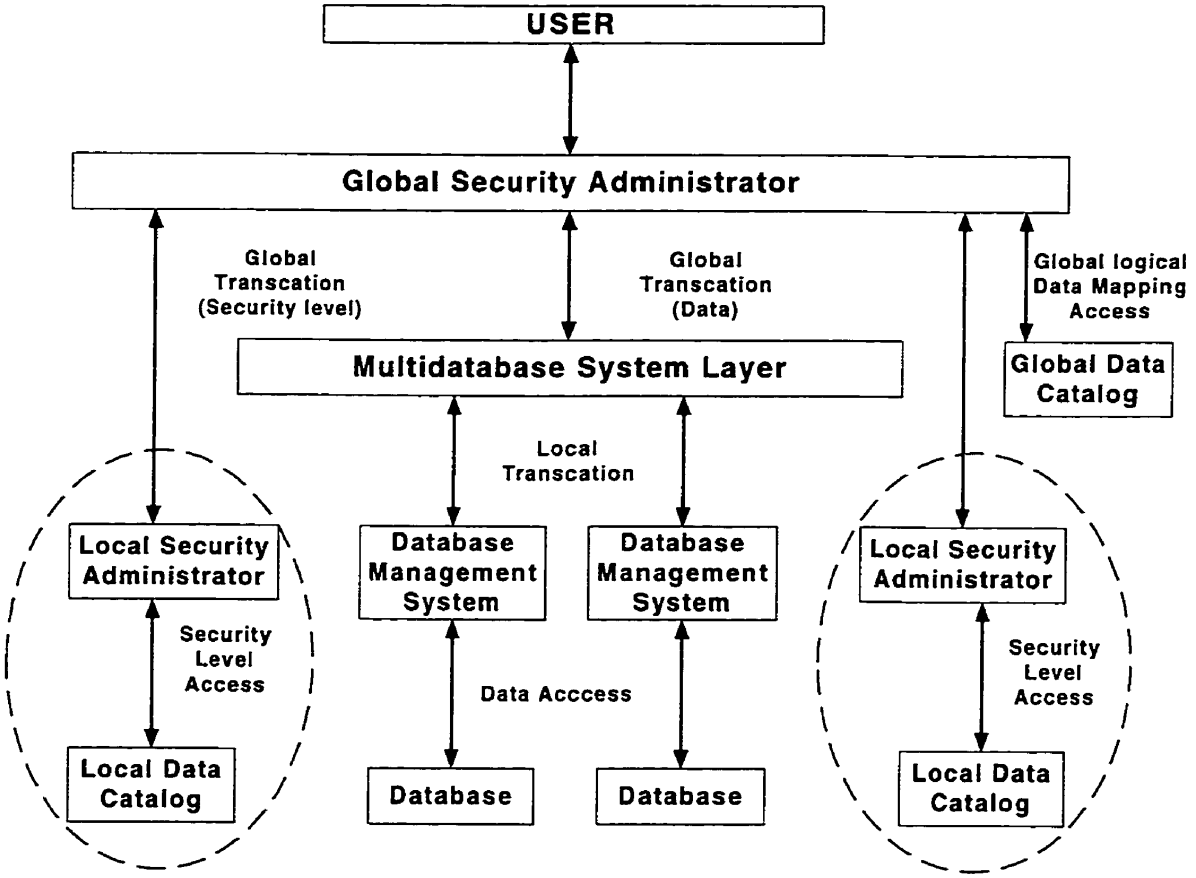

Figure 3.1 Secure Architecture Design For Multidatabase

Figure 3.1 illustrates four security components integrated into the MDBS architecture.

These are the global security administrator, local security administrator, global data catalog and local data catalog. The main functions of these security components are discussed below. One group of security components handles the interaction with users and another deals with data and security level storage and access.

The first group consists of the global and local security administrators. The second group consists of the local and global data catalogs. The following sections explain each component in detail.

3.1.1 Global Security Administrator (GSA)

The global security administrator is responsible for providing global user authentication, multilevel security mapping between the data/security levels, and global data filtration based on user security level for all the global transactions. The global security administrator accesses the global data catalog for the security level of global users, the type of operations user can perform (example: read, write and execute) on the database, and the security policies. Once the user's identity is confirmed, the user is authorized to send queries and access the data whose security level is equal to or less than the security level of the user. For example, if a user has a security level *Secret* he can access data that has security levels *Secret*, *Confidential* and *Non-Confidential.*

The global security administrator also performs multilevel security mappings between the data and the security levels. The global security administrator receives the query sent by the user. It checks if the query creates any table, inserts, or updates any data. If it does then the global security administrator inserts the corresponding security levels in the local data catalogs. The GSA then creates a logical mapping between the security levels and the data in the local databases and stores the information in the

global data catalog.

The GSA also performs data filtration, when the MDBS layer and the local security administrator return the results to the global security administrator after executing user query. One result contains the data and other contains the security levels of the data. After obtaining the results, the global security administrator gets the logical mapping between the security level and the data from the global data catalog. After retrieving the logical mapping, data filtration is performed based on the security policies and the result is displayed to the user. Figure 3.2. illustrates the information contained in the global data catalog in detail. Other functions of the global security administrator include:

- Checks for security policy violations;
- Provides audit records for different security events and processes that take place in the system;
- Updates information in the global data catalog.

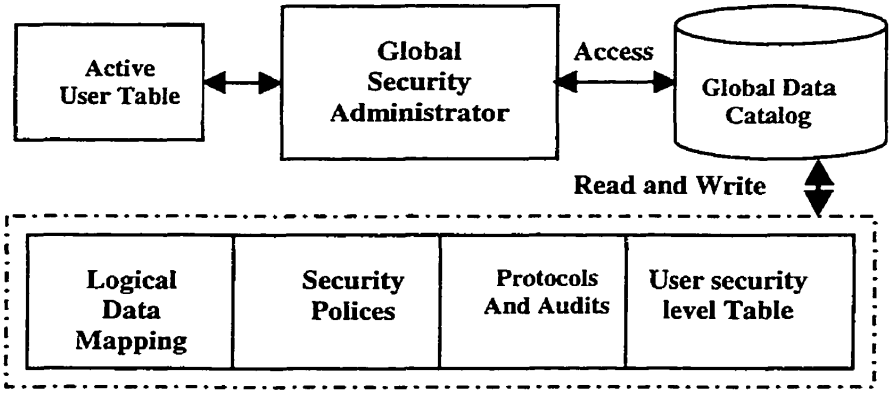

Figure 3.2 Global Security Administrator Design

3.1.2 Global Data Catalog (GDC)

The GDC is defined by integrating the local data catalogs at the participating site that forms the MDBS. The design of the GDC is similar to designing the global

conceptual schema in the multidatabase system, which involves the integration of either the local conceptual schema or a local external schema [32]. The GDC uses the local data catalogs to create a global schema for the MDBS. It contains information regarding user security levels, logical mapping of data and security levels, the operations that can be performed on the data, and user security level table. It also contains the security policies related to the use of network resources, database connections, and database accesses. Thus, every time a user sends a query, information in the GDC is updated.

3.1.3 Local Security Administrator (LSA)

The LSA provides local user authentication, local multilevel security mapping, and local data filtration for all local transactions in the local database system. The LSA also provides access to the local data catalog which stores the security levels of local users and maintains the multilevel security information present in the local data catalog. The local security administrator uses information from the user security level table present in the local data catalog for local data filtration. Other functions include:

- Authorizing the local DBMS for query processing;
- Checking for local security policy violation;
- Providing audit record for events and processes that take place in the local system;
- Updating information related to the local security level in the local data catalog.

3.1.4 Local Data Catalog (LDC)

The local data catalog contains all the information regarding the local users security level, operations that can be performed on the local data, and multilevel security tables from the local database. It also contains the security policies related to the use of local network resources. Local data catalog provides information to the global data catalog regarding local users and security levels.

3.2. Query Processing in SSA

Query processing techniques in SSA are not significantly different from query processing in any MDBS except at the query decomposition and execution stage. To simplify the description of query processing in SSA, the queries are divided into two types: Global User Queries (GUQ) and Local User Queries (LUQ). Both query types follow the three main query-processing steps, including *decomposition*, *optimization*, and *execution* [32].

3.2.1. Global User Query

A Global User Query (GUQ) involves two main stages of processing: global query processing and data filtration, as illustrated in Figure 3.3.

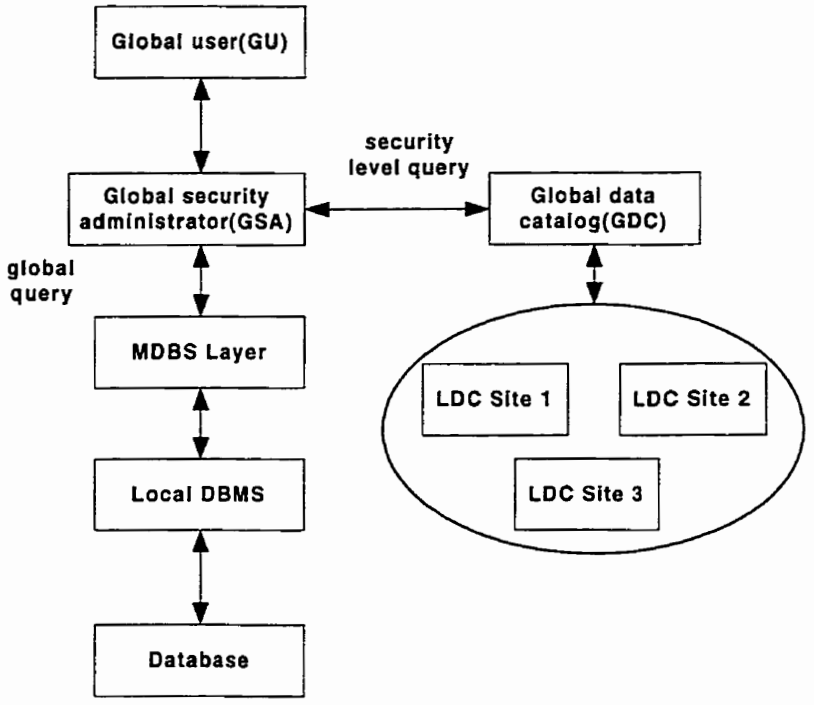

Figure 3.3 Global query processing

In the SSA system the security administrator at a site receives the query. First, a copy of the query is made. The original query is submitted to the MDBS layer and the

second copy after some modification is submitted to the GSA. Thus we have two queries: the original query to the MDBS is called Query_MDBS and the copied query is Query_GSA. Query_MDBS is split into subqueries based on data distributed across multiple sites. The only information that is required is the data allocation information that is stored in a global directory for the MDBS query. For the Query_GSA, the GDC stores the data allocation and the security level mapping information. Each subquery is then sent to the site where it is to be processed. The multidatabase layer at each site further "fragments" the query for each DBMS that it controls. At this stage the information within the global directory in the GDC is used. Each subquery is then translated into the language of the respective DBMS [32].

The query submitted to the individual DBMSs is processed and the results obtained from the local DBMSs are sent back to the global security administrator. The GSA stores the results in the cache. The Query_GSA is sent to a parser function. The parser function parses the query and modifies it so that the query can retrieve the security levels from the local data catalog. The modified query is then submitted to the local security administrators distributed across different sites, which are responsible for the query processing and retrieval of the required results. Once the results are retrieved they are integrated and sent to the GSA cache.

After the GSA receives the two results, it sends the results to the data filtration routine, which performs a logical mapping between the two results and retrieves the security level of the user from the active user table. The results are then filtered based on the user's security level. Only information that has the security level equal to or less than the security level of the user is displayed by the GSA to the user. The remaining results are hidden from the user. See Figure 3.3 for details.

3.2.2 Local User Query

Local user queries also involve two stages: local query processing and local data filtration as illustrated in Figure 3.4. In the local query processing stage, the local security administrator receives the query and makes a copy. One of the queries is sent to the local database layer for processing. The other query after some modification is sent to the LSA at the local site for processing. The local query processing stage in the local database includes decomposition, optimization, and execution [32]. The decomposition step involves the simplification of a user query that is specified in some relational calculus and its translation into an equivalent relational algebra tree over the conceptual schema. As illustrated in the example below [32].

"Find the name and address of all employees who work for the "research" department"

FNAME char(30)	*LNAME char (30)*	*DNO int*	*DNAME char(30)*	*ADDRESS varchar(30)*
John	*smith*	*5*	*Research*	*731 silver ave, TX*
Sam	*jose*	*6*	*Admin*	*444 santo la av, TX*
funi	*samu*	*9*	*Marketing*	*885 sibo av, TX*
dang	*dope*	*2*	*Admin*	*442 sarin rd, TX*

Relational Algebra:

RESEARCH_DEPT ← σ DNAME='Research'(DEPARTMENT)

RESEARCH_DEPT_EMPS←(RESEARCH_DEPT |X|DNUMBER=DNOEMPLOYEE)

RESULT ← Π FNAME,LNAME,ADDRESS(RESEARCH_DEPT_EMPS)

The optimization step involves reordering of relational algebra operations as well as determination of the best access paths to the data [32]. The resulting schedule is then executed by the run-time support system.

In the case of the LSA, before the query is passed on to the decomposition step, it is modified so that it can get the security levels present in the LDC. This is done using the parse function that helps in modifying the query. The query then goes through the three steps involved in the query processing, which are decomposition, optimization and execution. The results obtained from the query are stored in local cache of the LSA. In the local data filtration stage, the local security administrator retrieves the two results (data and security levels) from the local cache and performs a logical mapping between the results.

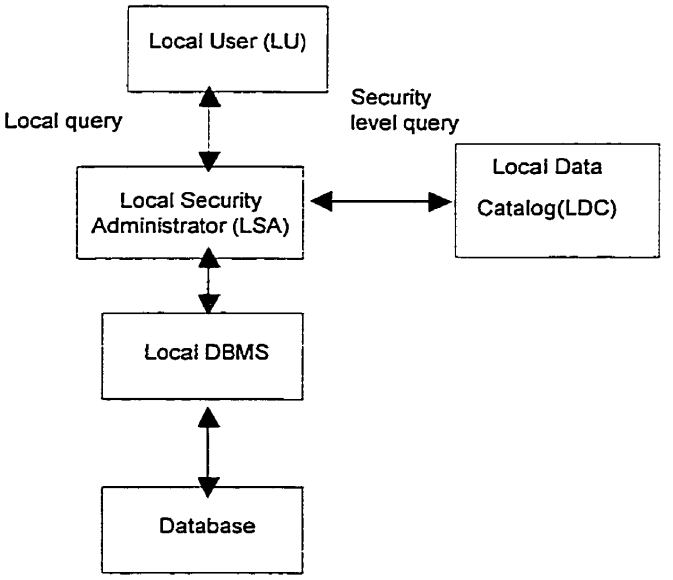

Figure 3.4 Local Query Processing

Then it filters the data based on the security level of the user obtained from the local active user table. Only data with security levels equal to or less than the security level

of the user is displayed by the LSA. The rest of the data is hidden from the user.

3.3 Algorithms and Pseudo code for the SSA

The following is the list of data structures and functions necessary to present the GUQ and LUQ algorithms.

3.3.1. Global User Query (GUQ):

- **Global_authentication (Global_User):** Global authentication function returns true if user is authentic otherwise it returns false.
- **Number_arguments (Global_Query):** Checks for number of arguments submitted by each global user.
- **Qcopy (S_Query,G_Query):** Makes a copy of the original query submitted by the global user. G_Query is the original query and S_Query stores the copy of the original query.
- **MDBS_layer (G_Query):** This function performs the multidatabase query processing and returns the results obtained from the G_Query.
- **GDC_layer (S_Query):** Global data catalog layer retrieves the security levels of the data that is returned by the G_Query.
- **GSA_Filter (Result_set_Data, Result_set_SecurityLevel):** Performs data filtration based on the security level of the global user obtained from the global user active table.
- **Logical_Mapping (Result_set_Data, Result_set_SecurityLevel):** Provides logical mapping between the data and the security levels.
- **Data_Filter (Result_set_Data, Result_set_SecurityLevel):** Filters the data based on the security level provided by the GSA_filter.
- **Display_user_results (Data):** Displays the results obtained from the GSA_filter to the global user who has submitted the query.

The pseudocode shown in Figure 3.5 has four stages.

Stage 1: Verification of user authentication.

Stage 2: Global user query processing and the global data catalog query processing.

Stage 3: Data and security level mapping and data filtration.

Stage 4: Display result set to the global user.

```
GUQ: Submission of a new Global_query (By Global_user)
     Input: new global user query to be submitted;
     Var
  G_User: Global user;
  Active_User_Table: Stores the active user list and their
                     security level;
S_Query: Temporary variable for storing a copy of the
         original query sent by the G_User;
G_Query: Global query submitted by the G_User;
     Output
          Result_set_Data: Data results submitted to the GSA;
Result_set_SecurityLevel: Data security level results
                          submitted to the GSA;

    begin                        /* STAGE 1 */
      if (Global_authentication(G_user) == false) then        ----1
          Return Login_Error;                                 ----2
      else                                                    ----3
             begin
             Active_User_Table(G_User+1);                     ----4
             if Number_arguments(Global_Query >= 1) then      ----5
           begin              /* STAGE 2 part a */
               for each (Global_Query to GSA) do              ----6
                 begin
                   Qcopy(S_Query,G_Query);                    ----7
                     Call Function MDBS_Layer(G_Query);       ----8
                     begin
                     Return Result_set_Data;                  ----9
                       if (Result_set_Data == true) then      ----10
                         Submit Result_set_Data to GSA;       ----11
                       else                                   ----12
                         Return Data_Error;                   ----13
                 end
                                 /* STAGE 2 part b */
                    Call Function GDC_Layer(S_Query);         ----14
```

```
                    begin
                    Return Result_set_SecurityLevel;          ----15
                    if (Result_set_SecurityLevel == true) then  -16
                        Submit Result_set_SecurityLevel to GSA; -17
                    else                                      ----18
                        Return Data_SL_Error;                 ----19
                    end
                              /* STAGE 3 */
                  for each G_Query to GSA do                  ----20
          GSA_Filter(Result_set_Data,Result_set_SecurityLevel); --21
        begin
         Logical_Mapping(Result_set_Data,Result_set_SecurityLevel);
                                                              ----22
         Data_Filter(Result_set_Data,Result_set_SecurityLevel); -23
                                Return Metadata;              ----24
                              /* STAGE 4 */
                    Display_user_results(Metadata Data)       ----25
                    end
              end
            end
             else
            Return Wrong_Number_Of_Arguments;                 ----26
end
```

Figure 3.5 Pseudocode for the Global User Query.

The Pseudocode for global user query consists of four stages. In stage one, the function *Global_authentication* (line 1) verifies and authenticates the global user identity and adds the global user into the *Active_User_Table* (line 4). If the global user authentication fails, the *Login_Error* (line 2) is returned.

Stage two contains two parts. In *part a*, the *Number_arguments* (line 5) function checks for the number of arguments submitted by the global user. If the arguments are more then one they are passed on to the global security administrator or an *Error* (line 26) is returned. The global security administrator, using the *Qcopy* (line 7) function, creates two copies of the user query. One copy of the query is sent to the *MDBS_Layer* (line 8) function to retrieve the data and in the *part b* the second copy of the query, after some modification, is sent to the *GDC_Layer* (line 14) function to

retrieve, the security levels of the data. After processing the queries the result set are stored in the cache of the global security administrator.

In stage three, the global security administrator processes the result sets obtained from stage two. It passes the two result sets (data and security levels) to the *GSA_Filter* (line 21) function for processing. The *GSA_Filter* function retrieves the logical mapping information from the global data catalog and calls the *Logical_Mapping* (line 22) function. The *Logical_Mapping* function establishes a logical mapping between the two result sets and passes the information to the *Data_Filter* (line 23) function. The *Data_Filter* function is responsible for retrieves global user security level from the *Active_User_Table* (line 4) and performs the data filtration process based on the security polices and global user security level. After filtration the metadata (result set) is returned to the GSA. In stage four, the data is retrieved from the GSA with *Display_user_results* (line 25) function and the results are displayed to the global user.

Let us consider the pseudo code's functions one by one in detail. The *Global_authentication* function from Figure 3.5 (line 1) accepts userid and password as arguments. When a user enters these two arguments the global security administrator starts a thread which verifies the information from the global data catalog. The Global data catalog stores all the information regarding the user security levels, logical mapping between the data and the security levels, global data directory information, security policies, and audit records. If the user is authentic the thread returns null values and starts up active user table thread or a login error is returned. The user is prompted to reenter the userid and password. The authentication process is described in detail in Figure 3.6.

The *MDBS_Layer* and the *GDC_Layer* function in Figure 3.5 (line 8 and line 14)

processes the original query sent by the user to retrieve the data from the databases. These functions perform the five important processes needed to process the query and retrieve the data from the databases. These functions are query processing, transaction management, scheduler, recovery management and runtime support processor [11]. The result sets obtained from the process are stored in the cache and the connections to the databases are closed. The *GDC_Layer* also performs the same processes as the *MDBS_Layer*, but with respect to the shadow database. The shadow database stores the security levels of the data. The results returned from the processes are stored in the cache. These two processes are show in detail in Figure 3.7.

The *GSA_Filter* function in Figure 3.5 (line 21), retrieves the two result sets from the cache and starts processing them. It retrieves the logical mapping between the two result sets from the global data catalog and maps the data with the security level. After mapping is done it starts the data filtration process. In this process the global user's security level and security policies are retrieved from the user security level table in the global data catalog. After that data is filtered based on the security level of the user. Data that has security level equal or less than the security level is retrieved from the result set and is passed on to the global security administrator is displayed to the user with the *Display_user_results* function in Figure 3.5 (line 25). See Figure 3.7 for details.

3.1.2 Local User Query

Local user queries have the same four stages as described above. That is authentication, query processing, data filtration, data mapping and displaying the results to the local user. The only difference between the two is the components that process the two queries (i.e, L_Query and the S_Query), where the L_Query is the local query and the S_Query is the copy of the L_Query. Instead of the multidatabase

layer and global security administrator, it is local DBMS and local security administrator (recall Figure 3.4).

3.4 Comparison between SSA and other Security Approaches Discussed

In Section 2.4, three types of architecture models are discussed: the kernelized architecture (partitioning architecture), the replicated architecture, and the encryption model. In this section these three architectures are compared with SSA. In case of the kernelized architecture, each security level is partitioned into a separate database and the user uses a separate DBMS for each security level to manage data at or below that level.

In the SSA, there are only two partitions: the local database, which stores the data, and the data catalog, which stores the security levels for the data. The SSA approach solves some of the disadvantages that exist in the Kernelized architecture, including redundancy and accuracy. Because there are only two partitions in the SSA, there is less management and replication of data.

The kernelized architecture does not address the problem of high-level users, who needs to access some low-level secure data that must be combined with high-level secure data. The SSA solves this problem because high and low-level secure data are not located in separate databases and they are provided to users based on their security level (using data filtration).

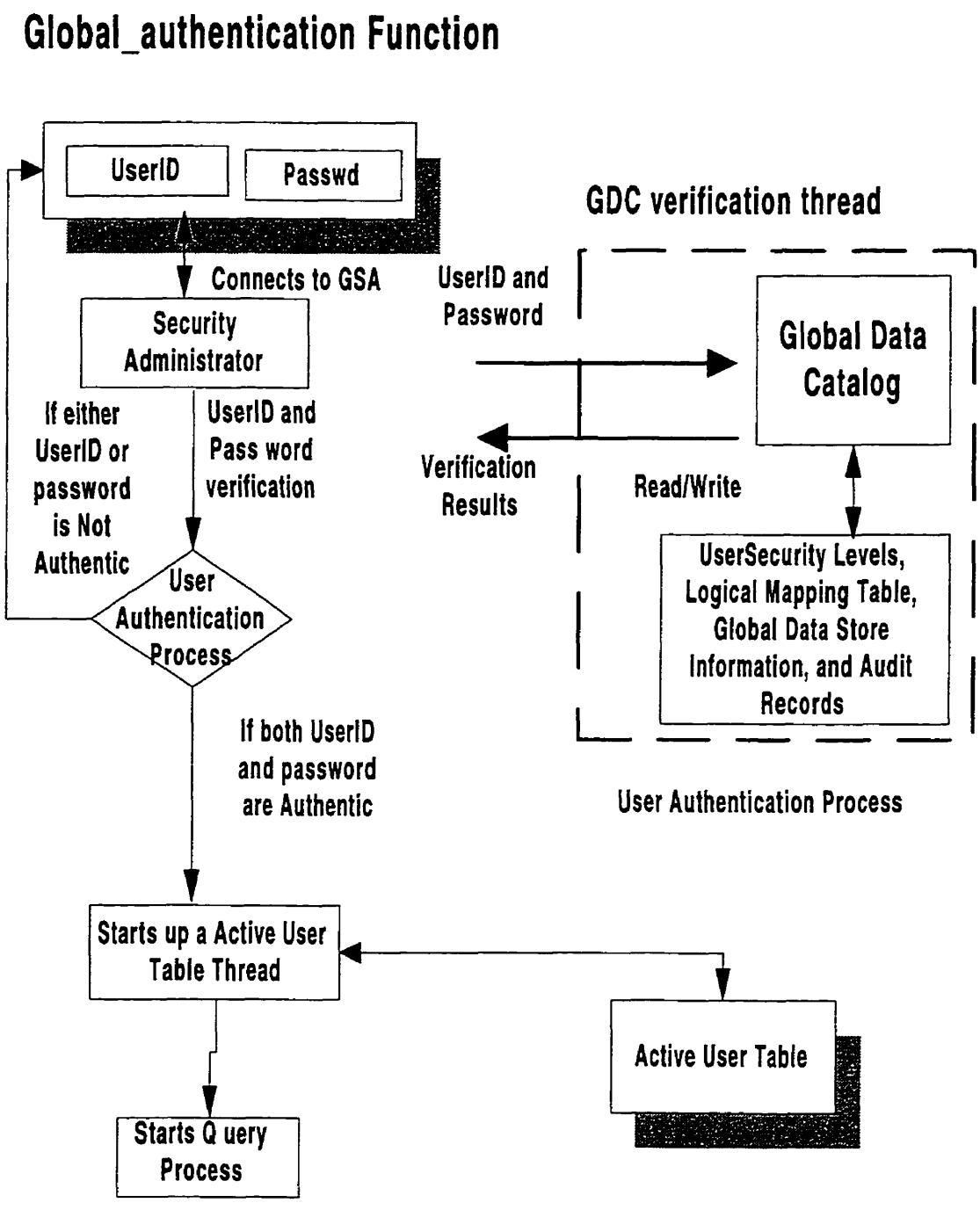

Figure 3.6 Detailed description of the user authentication process.

MDBS_Layer and GDC_Layer functions

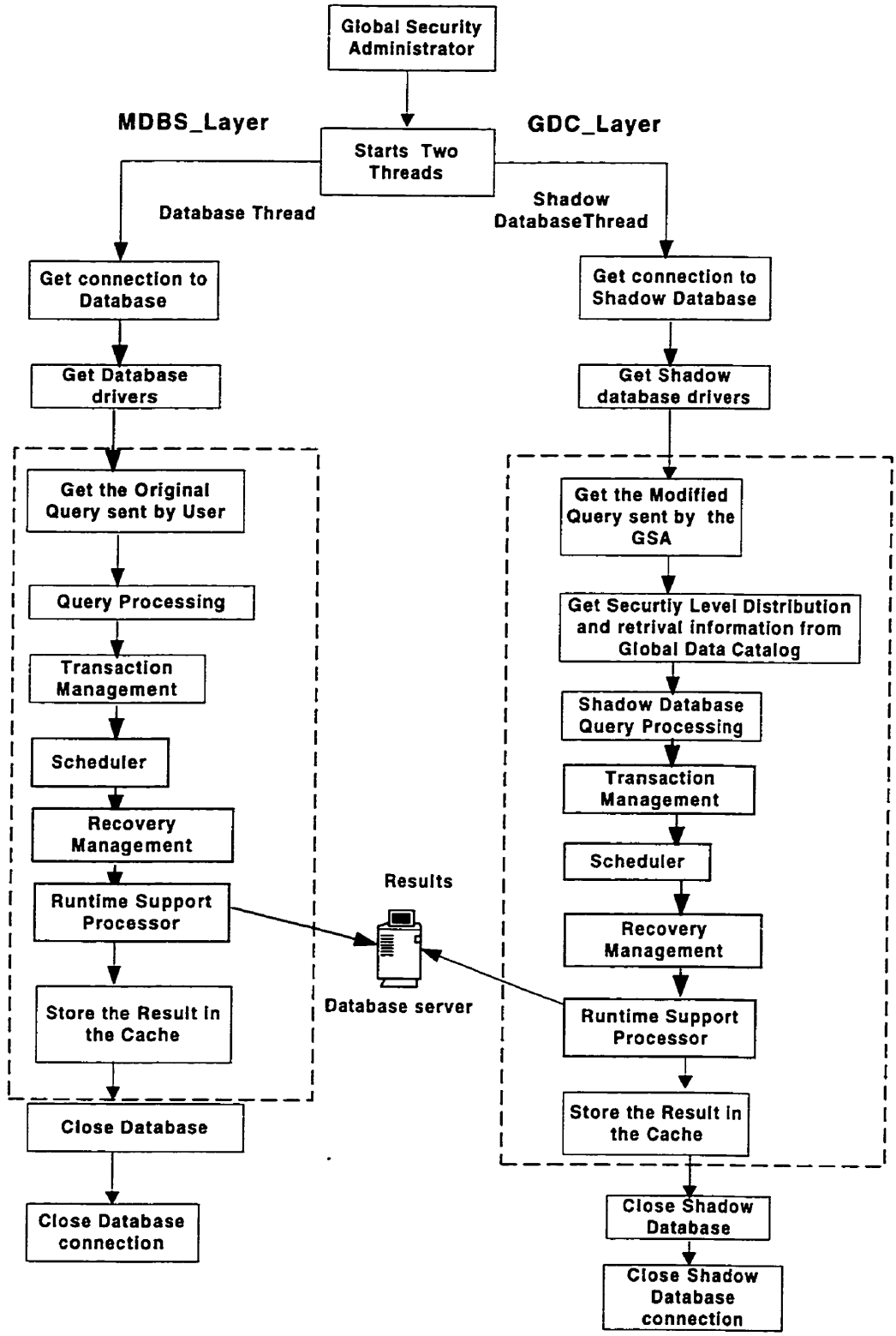

Figure 3.7 Explains the MDBS_Layer and the GDC_Layer Function

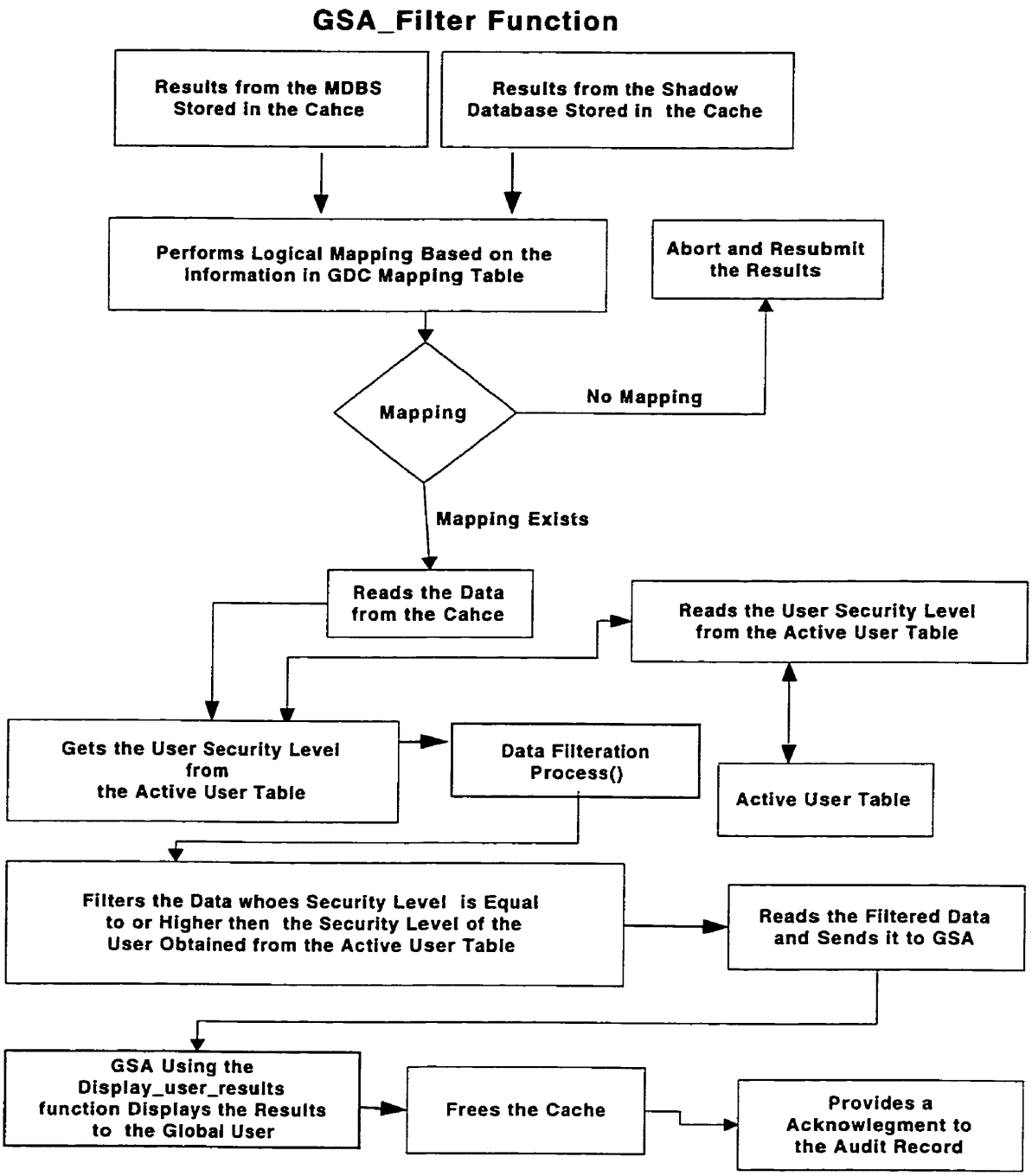

Figure 3.8 Flow Chart of GSA_Filter Function

There is one noteworthy advantage to the kernelized architecture over the SSA. The scheduler that is responsible for concurrency control is separate for each DBMSs so the scheduler can be implemented with untrusted code, that is, not part of the TCB[1].

The replicated architecture is similar to the kernelized architecture. The difference lies in storage of each security level data in separate containers. The problems encountered by the replicated architecture are similar to the kernelized architecture and are addressed as mentioned above.

The encryption model uses different encryption schemes for different security levels. If a user accidentally receives data, which has higher security level, it will be unreadable. This model is one of the most secure models and solves most of the problems encountered in the previous two architectures. Two problems persist in processing of data and users mounting a plain text attack because the data is visible to the user. In the SSA there are two reasons because of which users getting higher security level data is reduced greatly. First, the data and the security levels are logically, not physically, linked with each other. Second, because of data filtration process the user can get only data that has the security level equal to or less than the security level of the user. But there exists a problem of data available in plain text in the SSA. Encrypting the data can solve the problem of plain text data, but that is beyond the scope of this thesis.

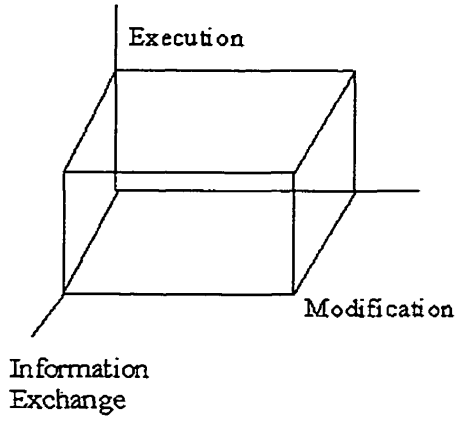

Figure 3.9 Autonomy Dimensions

The other security approaches and problems that we have tried to solve using the SSA

are, the cascade problem, transparency, and multilevel-security model. These are discussed in Section 5.3.

3.5 Autonomy Requirements and Autonomy Violations in SSA

Autonomy is an important factor by which a MDBS is characterized [2]. Because the SSA design exists on top of MDBS we have to consider the implications of autonomy on the SSA and what autonomy violations are acceptable in order to provide a secure MDBS. In other words autonomy provides us a means of understanding the degree of transparency provided to the local database systems in the SSA.

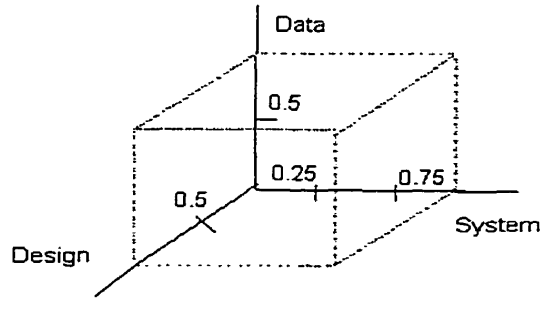

Figure 3.10 Modification Dimension

There are many research papers that describe autonomy in multidatabase system [2,3]. But defining autonomy is subjective and system dependent. To provide a description of autonomy that covers all the aspects of a system is difficult but if we consider it as a dimension in space it can be considerably simplified. In this thesis, autonomy is composed of three major dimensions [2,3]: modification, execution, and

information exchange as illustrated in Fig 3.9. Modification dimension deals with the modifications necessary for a local DBM to participate in a MDBS.

Execution dimension deals with which level has the control of execution of transactions or operations on the local DBMS.

Finally, the information exchange dimension address the amount of information that must pass between the local and global levels to permit a DBMS to join or leave the MDBS and to ensure correct execution while it is functioning [2].

Considering the above dimensions we can evaluate the SSA. Let us consider Figure 3.10 for evaluating the modification dimension. In this figure the initial point represents no autonomy violation. Each axis is of equal length and the maximum violation is one. The first axis considered is that of modification to the system [3]. The SSA requires additional software that is not part of the DBMS (local security administrator and local data catalog) so this violation is considered 0.25 mark on the system axis. The second axis is the data modification axis, where there are some modifications made to data stored in the database. In SSA, no such modifications are required. This axis is marked at 0. The design modification axis addresses changes to the underlying local schemas used at each DBMS [3]. SSA does not provide enough data to mark this axis.

Next, we will consider execution dimension, which deals with autonomy violations

related to the control of transactions and their operations on the multidatabase system. The execution dimension axis is further divided into local and global transactions. In SSA, it is necessary to coordinate the execution of local transactions (local database) with the local data catalog (security level) transactions through interactions with the MDBS. This autonomy violation is 0.5 mark on the local transactions. There is no sufficient information to mark the global transaction axis.

The information exchange dimension deals with the amount of information traffic that occurs between the local database layer and the multidatabase layer. This dimension can be further divided into three areas: data information, execution information, and schema information. The SSA does not have sufficient information to mark the execution information axis. In the data information axis the SSA is at mark 0 since we assume that all the data in the system is relational. So there is no change of data format from local to multidatabase layer. In the last case, schema information axis, there is some schema information exchange between the local database, local data catalog and the global data catalog so mark at 0.5 in the schema information.

The total autonomy violation of SSA is calculated by considering all the three dimensions and the criteria defined in the technical report by Barker [2,3]. The final dimension of SSA are described below:

Modification Dimension	Axis Values	Execution Dimension	Axis Values	Information Exchange Dimension	Axis Values
System	0.25	Local Transaction	0.5	Execution	Not Applicable
Data	0.5	Global Transaction	Not Applicable	Data	0
Design	Not Discussed			Schema	0.5

The Quantification Measure [3] for the SSA is 0.9682.

Chapter 4

Implementation of the Secure Software Architecture

This chapter describes the implementation of the Secure Software Architecture (SSA). The provision of mechanisms to satisfy the U.S Department of Defense's evaluation criteria for a secure system is the main goal in the implementation of the SSA. The SSA was implemented on a relational database system for the purpose of testing, verification, and prototyping. We have implemented the functions performed by the local database system, local security administrator, and local data catalog as a part of the SSA.

4.1 System Overview

Figure 4.1 shows an overview of the prototype of the SSA system. The SSA was implemented using the local area network at TR*Labs*[1]. The system consists of a database server and a remote client, both running Windows 98. Connection between these two PCs is established using TR*Labs* web server. The database server contains the tables and the software needed to run the SSA.

[1] TRLabs is Canada's largest not-for-profit information and communications technologies research consortium and are internationally recognized as a leading model for industry-university-government collaboration.

4.1.1 The Database

To test the prototype SSA, there must be a database that is able to answer queries sent to it over the network by a client. In the current system, these databases are modeled by the InstantDB system. InstantDB is a Java based Relational Database Management System (RDBMS).

InstantDB was chosen to model the LDBS, local data catalog, and the shadow databases because it has several desirable features:

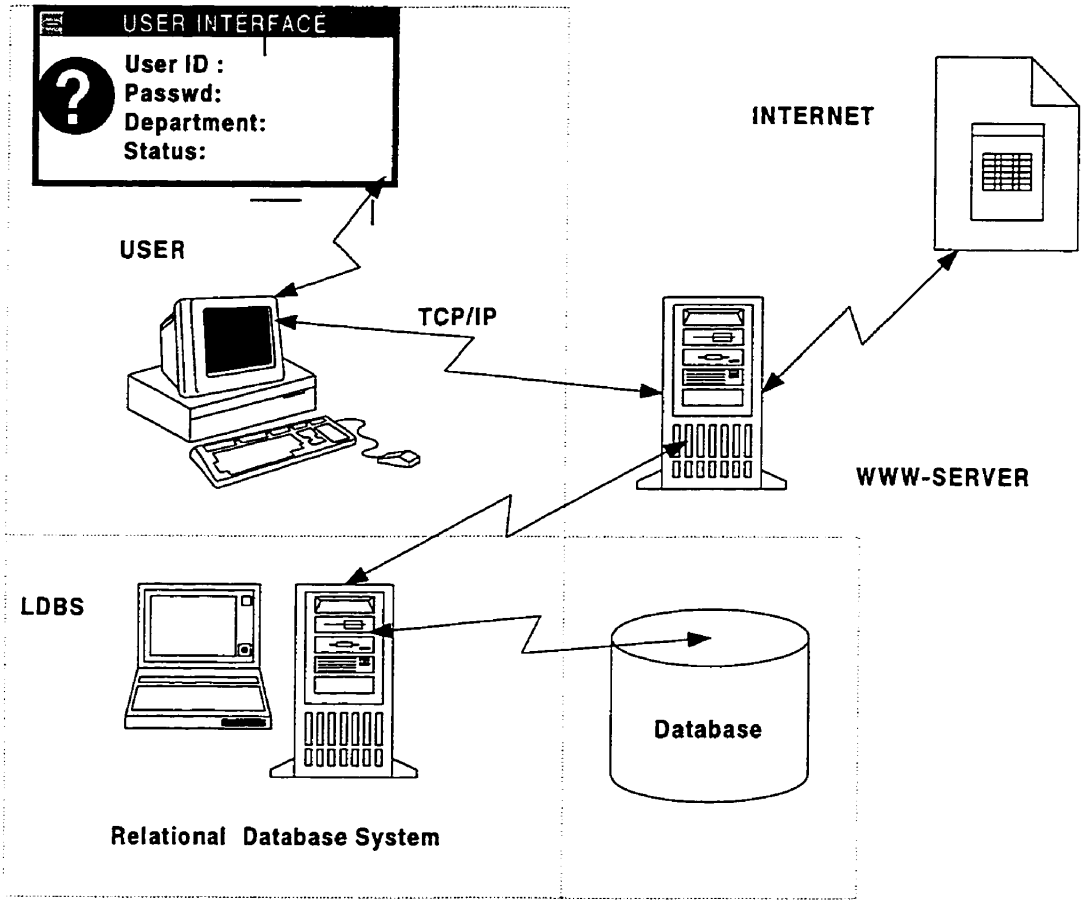

Figure 4.1 An Overview of the Secure Software Architecture System (SSA)

- *It is a relational database.* Relational databases are simple, easy to use and their modeling abilities are sufficient for most business applications that are in widespread use.
- *Remote queries.* It can act as a server in a client/server scenario. This is helpful because LDBS and the shadow database can be treated as two independent databases and be accessed using two separate connections, which provide faster access and a better data filtration process. It also maintains autonomy over the LDBS data and database.
- *Multi-user capability.* Up to 25 users can use the database engine concurrently, which can be further increased with a faster machine. It is also free for academic use.
- *Portability:* By using standard SQL and Sun's JDBC API, InstantDB ensures that the applications enjoy a high degree of portability.
- *InstantDB is accessed using the JDBC driver.* The JDBC API provides universal data access from the Java programming language. Using the JDBC API, we can access virtually any data source, such as relational databases, spreadsheets, and flat files.

4.2 Implementation Language Used

The implementation of SSA consists of two parts: a front-end process and a back-end process. The user interface and the security administrator perform the front-end process. The back-end process is preformed by the local database system (InstantDB)

and the local data catalog. To develop and provide information transfer between the front and back ends, Java[2] programming language was used.

A single SQL statement can be very expressive and can initiate high-level actions, such as sorting and merging data. Unfortunately, a user must connect to a database before executing the SQL commands, and each database vendor has a different interface, as well as different extensions of SQL.

For example ODBC is a C-based interface to SQL-based database engines, which provides a consistent interface for accessing the metadata (information about the database system vendor, how the data is stored, *etc.*). Individual vendors provide specific drivers or "bridges" to their particular database management system. Consequently, with ODBC and SQL, we can connect to a database and manipulate it in a standard way. Though SQL is well suited for manipulating databases, it is unsuitable as a general application language and programmers use it primarily as a means of communicating with databases. Thus, another language is needed to feed SQL statements to a database and process results for visual display or report generation.

Unfortunately, one cannot easily write a program that will run on multiple platforms even though the database connectivity standardization issue has been largely resolved. For example, a database client written in C++, would have to be totally

[2] Copyright 1993-1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California, 94303, U.S.A. All Rights Reserved.

rewritten for each platform; that is, the PC version database client would not run on a Macintosh. A Java program can run on any Java-enabled platform. The Java language is completely specified and, by definition, a Java-enabled platform must support a known core set of libraries. One such library is JDBC, which can be thought of as a Java version of ODBC. Using Java in conjunction with JDBC provides a truly portable solution to writing database applications.

4.3 Data Structures

The front-end process implements the user interface and security administrator. Back-end process consists of local database, shadow database, and data filtration process. The front-end is described first.

1. *User interface*

The user interface is used to enter data and other information to be processed. The *SSA_MDBS* class is the main class in the user interface. It is responsible for starting up the user interface. It has fields for the Userid, Password, User Department, and Login Status. It also contains methods that process the information that is provided by the user. The *SSA_MDBS* class contains two main methods: `Action` method and `Contains` method. The following is a skeletal interface of the *SSA_MDBS* class.

```
Syntax:
Class summary:
    public class SSA_MDBS extends JFrame( )

Constructor summary:
    public SSA_MDBS( )

Method summary:
```

```
public boolean Action(Event actionevent, Object args)
public void windowClosing(WindowAdapter() close )
public boolean Contains(String userid, String password)
```

The *Action* method looks after the actions performed by the user. For example, when the user enters the USER ID, PASSWORD, DEPARTMENT NAME, and presses the OK button, the *Action* method is invoked. The *Action* method gets the information in the form of two parameters, Event and Object. Event can be the user pressing submit, cancel, or clear button. Object is the combination of USER ID and PASSWORD.

The *Action* method invokes the *Contains* method by passing the USERID and PASSWORD. The *Contains* method is used to verify the login authenticity of the user. It is also responsible for starting up the Active User Table, which stores the USER ID and security level. If the user is genuine, it will close the user interface screen, display the user validation, and start up the security administrator thread. Otherwise, it refreshes the user interface screen, tells the user to re-enter the USERID and PASSWORD, and records the event in the Audit Record.

2. Security Administrator (SA)

The SA is implemented by the *dbengin* class. This class is responsible for displaying the SQL query user interface and connecting from the interface to the local database and the shadow database. The shadow database contains all the security levels of the data stored and is used for logical security level mapping. The following is a skeleton class and its methods.

```
Class summary:
    public class dbengin extends Frame implements
ActionListener, WindowListener( )

Constructor summary:
    public dbengin(String title, String propfile)

Methods summary:
    public boolean dbengin(String title)
    public connectToDB( )
    private void closeDB( )
    private void closeStatement ( )
    private void copyresultstomem( )
    private void executeSQL ( )
    public void actionPerformed( ActionEvent event)
    public void windowsClosing(WindowEvent e)
    public void dispose( )
```

Once the user authenticity is established, the *dbengin* class is started. The *dbengin* class invokes the method *dbengin(String title)*. This method gives a title to the SQL graphical interface and display. It also initializes all events and waits for an action event from the user.

The user enters the SQL query in the SQL query text field and presses the submit button. The submit button triggers an action event and invokes the *actionPerformed (ActionEvent event)* method, which passes the event parameter to the object function that starts up the *ConnectToDB( )* method.

The main function of the *ConnectToDB( )* method is to start up two threads. One thread is responsible for initializing, retrieving, and checking the location of the database and the JDBC driver to connect to the database. The second thread is responsible for initializing, retrieving, and checking the location of the shadow

database and the JDBC drivers required to connect the shadow database. After the information is retrieved and processed, connections are established to the local and shadow databases. The data from these two databases are used to identify the security level of the data.

Let us consider an example. When a user creates a table in the SSA, he sends the following query.

CREATE TABLE EMPLOYEE_INFO(Name varchar(30), EmployeeID int)

The security administrator makes a copy of the original query and sends the query to the MDBS layer. Then the copy of the original query is modified such that it can create a similar table in the shadow database. The modified query is as follows.

CREATE TABLE EMPLOYEE_INFO(Name_SL varchar(10), EmployeeID_SL varrchar(10))

This query creates a table with field *Name_SL* and *EmployeeID_SL* in the local data catalog. When users send queries requesting information from the *EMPLOYEE_INFO* table the security administrator again makes a copy of the original query and sends the original query to retrieve data from the databases. Then it modifies the copy of the original query to retrieve the security levels from the data catalog. When users want to insert information into the tables. The original query inserts the data into the table and the modified query inserts the security levels of the user obtained from the active user table instead of data.

Insert into Employee_Info value("JOHN",64945676) (Original query)

Insert into Employee_Info value("Top-Secret","Top-Secret") (Modified query)

Let us consider an example where user's with security levels *Top-Secret* sends a query in the following format.

Select Name, EmployeeID From Employee_Info (Original query)

Select Name_SL,EmployeeID_SL From Employee_info (Modified query)

There are two tables named ***Employee_Info*** within the system, one in the local database and the other in the shadow database. Finally, the format of the tables are as following.

Employee_Info *in the Local Database:*

Name	***EmployeeID***
JOHN	**64945676**
TOM CRUSE	**85497405**
HARISON FORD	**82151945**

Employee_Info *in the Shadow Database:*

Name_SL	***EmployeeID_SL***
Top-Secret	**Top-Secret**
Secret	**Secret**
Classified	**Classified**

If user with Top-Secret sends the above select query, all data is visible. This means all the rows will be returned for the above query.

In the next step, the *dbengin* invokes the *ExecuteSQL( )* method. This method retrieves the SQL query that is presented in the text field of the graphical user interface and passes the query to the local and shadow databases. The local and shadow database processes are back-end processes that will be explained later in this section.

The local and shadow databases process the SQL query and send the results to the *Copyresulttomem( )* method. This method receives the two results, analyzes and copies the information to two different locations in memory and sends a signal to the security administrator (SA). d*bengin* class closes the database, closes the two connections and invokes the *dispose( )* method which refreshes the graphical user interface for the user to enter a new SQL query. The results are displayed to the user after data filtration process completes.

3. *Local Database System*

The Local database system is responsible for storing and managing user databases. When the e*xecuteSQL( )* method from the *dbengin* class passes the SQL query to the Instant database system, the instant database system goes through different stages which are described below using an example.

A *Statement Method* (a Java method in the SQL package) executes the static SQL statement and obtains the results produced by it. This is done using the classes and methods described below.

```
public class DriverImpl implements java.sql.Driver {}
   static {
         try {
       Class cls = Class.forName("DriverImpl");
    java.sql.DriverManager.registerDriver((java.sql.Driver)cl
s.newInstance());
     } catch(Exception e) {
            e.printStackTrace();
                    }
}
```

The *DriverImpl* class gets the required drivers to communicate with the InstantDB system. When a driver class is loaded, it creates an instance of itself and registers itself with the *DriverManager*. The static initializer registers the driver in the JDBC *DriverManager*. This means that a user can load and register a driver by executing *class.forName("foo.bah.Driver")*. A driver converts the JDBC SQL grammar into its system's native SQL grammar prior to sending it to the database using the method:

```
public String nativeSQL(String sql) throws SQLException {}
```

The *nativeSQL* method returns the native form of the statement that the driver would have sent:

```
public class ConnectionImpl implements
java.sql.Connection {}
```

The *ConnectionImpl* class represents a session with a specific database. Within the context of a Connection, SQL statements are executed using the *executeQuery* method and results are returned.

```
public ResultSet executeQuery(String sql) throws SQLException {}
```

The *executeQuery* method receives the SQL expression and executes the query. Once the execution is performed it goes to the next stage of processing. To retrieve

the result set from the database. This is done using the class *getMaxFieldSize()*.

```
public int getMaxFieldSize() throws SQLException {}
```

The *getmaxFieldSize* method gets the data for any column value (in bytes). The *getmaxFieldSize* method gets column values like binary, varbinary, longvarbinary, char, varchar, and longvarchar columns.

Once the query passes through the above stages, the results of the query are sent back to the *dbengin* class for further processing.

4. Shadow Database System.

The main functionality of the *shadow database* is to store the security levels for the data present in the local databases. Now let us consider how the query is parsed. When the *executeSQL( )* method sends the SQL query to the shadow database thread. The *ldc* class receives the query and identifies the type of SQL statement the user has sent. For example, Data Manipulation Language (DML) which can query and update the data, Data Definition (DDL) which defines the objects in a database, Data Control Language (DCL) which controls access to the data. After identifying the type of SQL statement, the *ldc* class invokes the *parsequery(String query)* method which takes the SQL statements and modifies it according to the user security level and security policies. After modifying the SQL statements the *ldc* class establishes a connection to the *shadow database*. It then submits the SQL query to the *shadow database* for processing. The *shadow database* processes the SQL

query and sends the results if any, in the form of parameters to the *resulttomem(resultset data)* method where the results are processed and copied to the memory location. The following is the skeleton of the shadow database:

```
Class summary:
public class ldc extends Thread( )

Constructor summary:
public ldc( )

Method summary:
public void run( )
public void parsequery(String query)
public getConnection( String path)
private static void resulttomem( resultset data)
```

5. Data Filtration

The main purpose of the data filter is to read the query results from the cache and filter it with respect to the user security level that is present in the active user table. After filtering the data, it displays the results to the user. The detailed process is as follows.

The *filter* class is responsible for performing the tasks described above. The syntax of the *filter* class is as following:

```
Syntax:
    Class summary:
              public class filter extends Frame( )
    Constructor summary:
              public filter(String title)
    Methods summary:
              public String StringTokenizer( )
              public void Securityl( )
              public void Displayr(Metadata data )
```

The `filter` class is invoked by the security administrator after the results from the two databases (shadow and local database) are stored in the memory by the `Copyresulttomem( )` method. The `filter` class reads the data from memory, then it performs a one-to-one mapping between the data stored in the two memory locations. After the mapping is established using the `StringTokenizer( )` method, it invokes the `Securityl( )` method which reads the user security level from the active user table. After the security level of the user is retrieved the data filtration process begins. The data filter replaces the data in the memory with a *"NULL"* wherever the security level of the user is less than the security of the data (i.e., if the user security level is 3 than the data with security level less than that is replaced with *"NULL"*). After performing the filtration of the data, the `filter` class invokes the *Displayr(Metadata data)* method. This method reads the data from the memory and displays the information to the user with the help of the Security Administrator. The security administrator discards the *"NULL"* values at the filtration stage. Thus, only information that has the security level equal to or less than the security level of the user is displayed. Now let us consider examples of the user interface and describe there functions used in the Secure Software Architecture.

4.4 User Interface (UI)

Users send queries using the UI and the results are displayed through the same interface.

4.4.1 Login Editor

The Login Editor (see Figure 4.2) is a tool the database administrator (human guard) uses to add or delete a user profile.

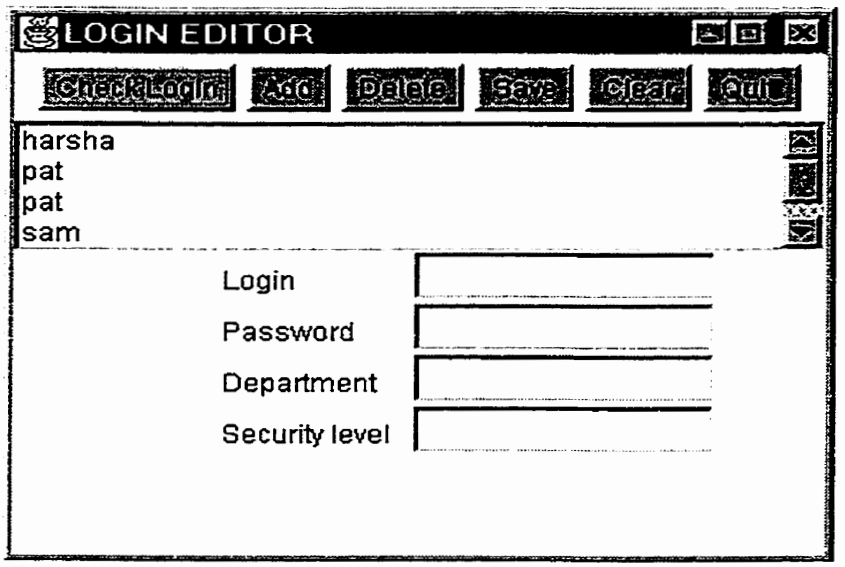

Figure 4.2 Login Editor

A user profile consists of *loginid, password, department, and security level.* To access the SSA system the user must have an entry in the Login Editor. The main function of the Login Editor is to process the user profile and store the information in the local data catalog. This information is used by the security administrator to authenticate a user loginid and password. It is also used for creating an active user table. The Login Editor consists of four text fields where the database administrator enters the loginid, password, department, and security level of the user. After entering the information, using the add button, the user profile is added. The CheckLogin button is used to check the loginid and password of a particular user.

4.4.2 Multidatabase System Login

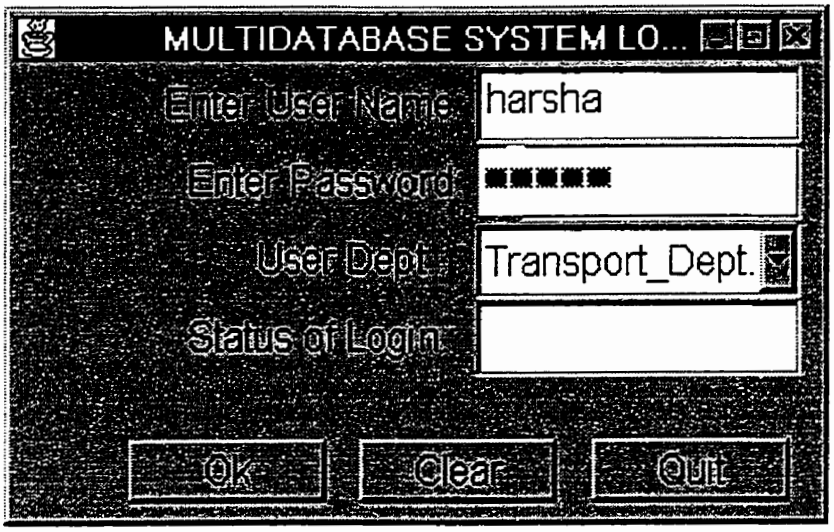

Figure 4.3 Multidatabase System Login

The Multidatabase System Login (MSL) user interface is used to enter the SSA system for users. Let us consider an example that illustrates the detail process from a user entering to the SSA system to when results are displayed to the user. In this process there are two users: user one ("harsha") has a security level 1 and user two

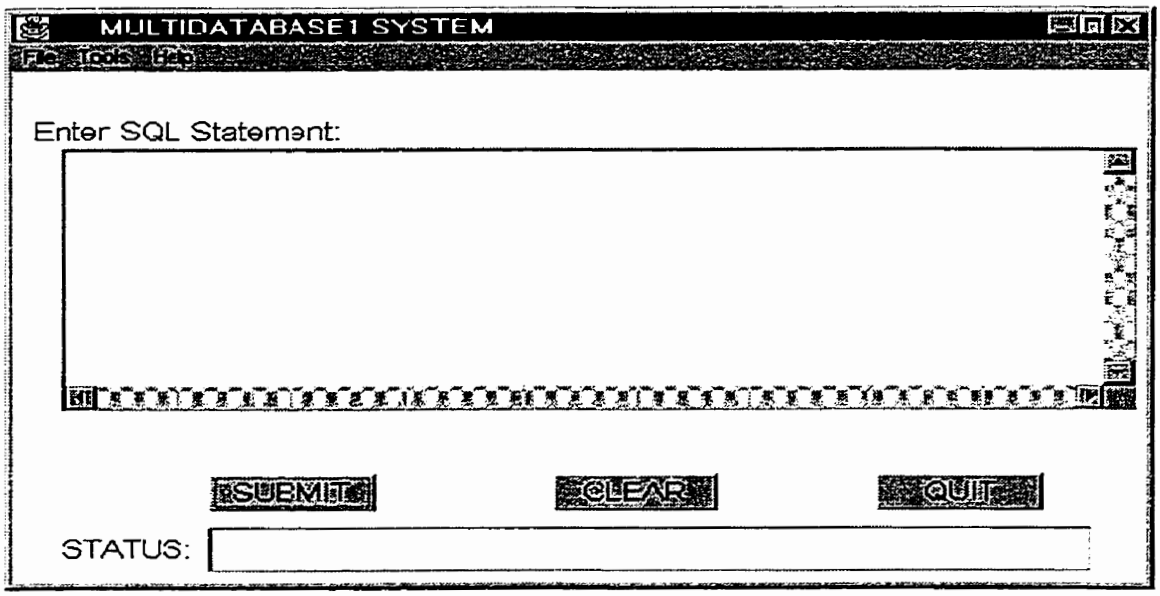

Figure 4.4 SQL Query User Interface

("pat") has a security level 4. User one enters the loginid and password as shown in Figure 4.3.

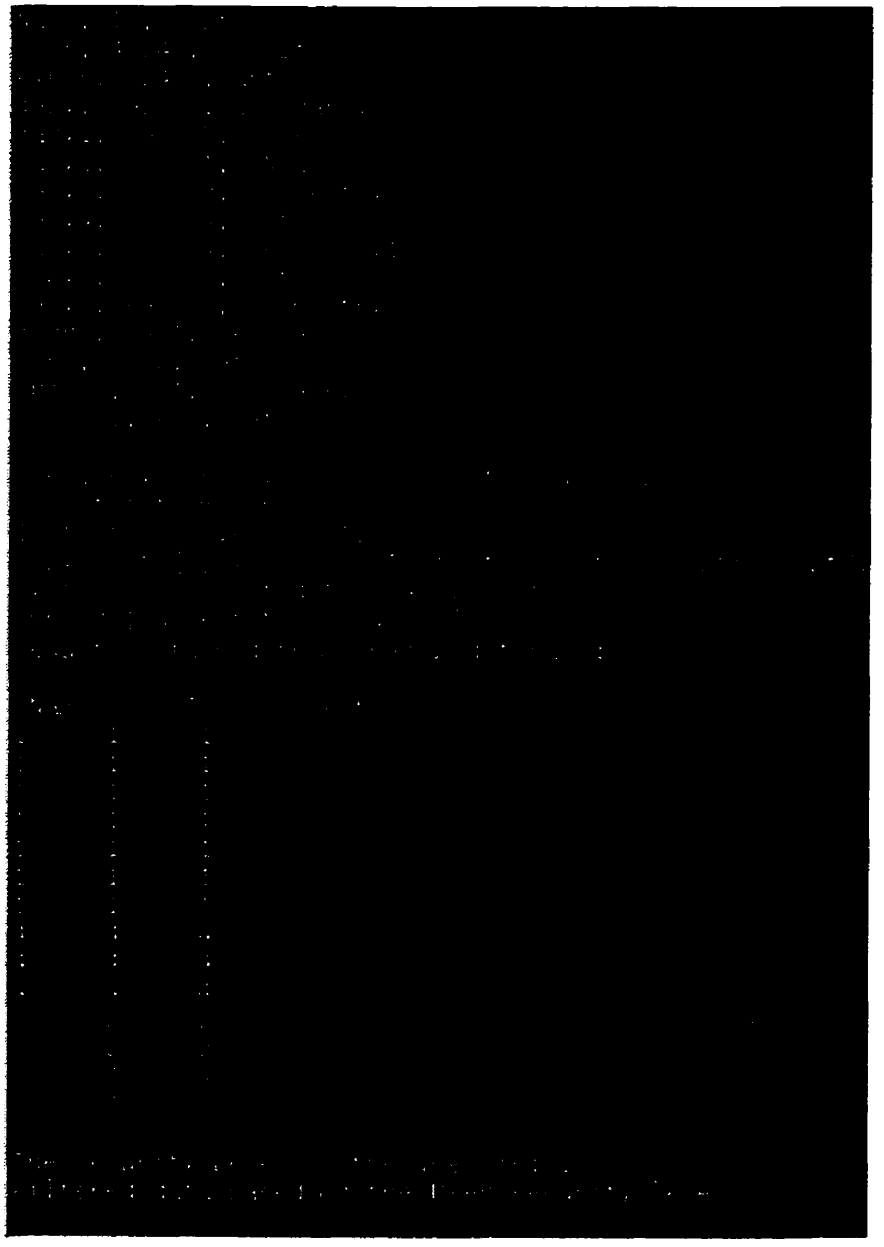

Figure 4.5 Information that is Recorded in the Audit Record (Security Level 1)

Once the user enters the information in the MSL, the system administrator performs user authentication using the information present in the local data catalog. After the

login verification, the SSA system starts up the SQL query user interface.

4.4.3 SQL Query User Interface

The SQL query user interface is responsible for providing a text field where the user enters the SQL query that is to be processed by the SSA system, as shown in Figure 4.4. The user with security level 1 enters a SQL query and presses "submit" button. The SSA system starts processing the query and records the information in the audit record, as shown in Figure 4.5. The next stage in processing is data filtration and results. As illustrated in Figure 4.5 the SSA system retrieves all the information that is stored in the local and shadow databases. It also retrieves user security level and the query sent by the user to the system.

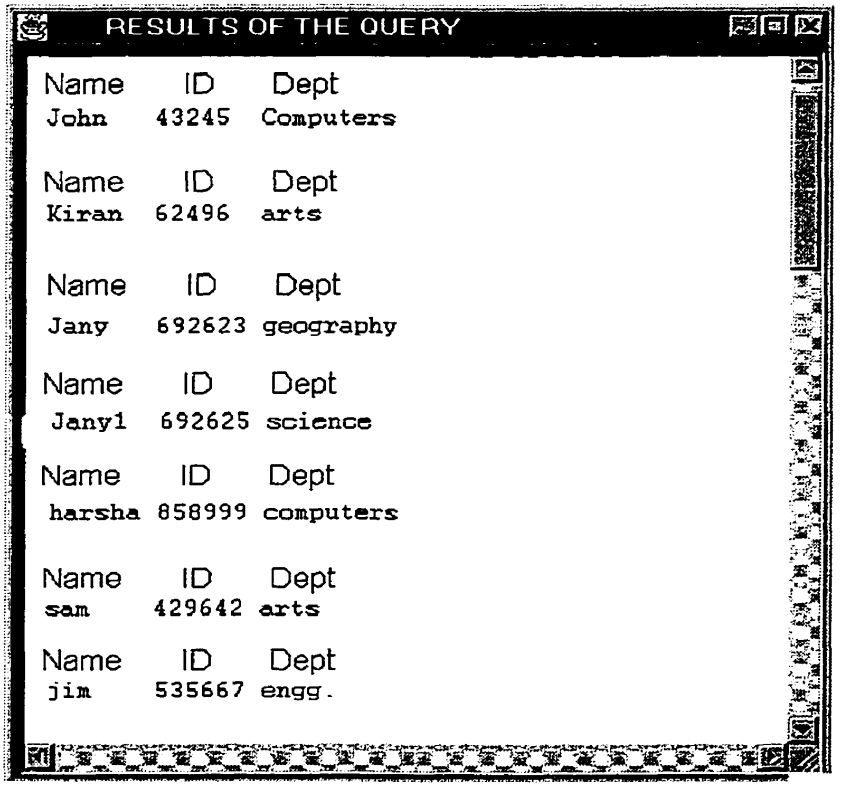

Figure 4.6 Results Obtained by User with Security Level 1

4.4.4 Results of the Query

It is a user interface that displays the results obtained from the query sent by the user to the system after data filtration is performed.

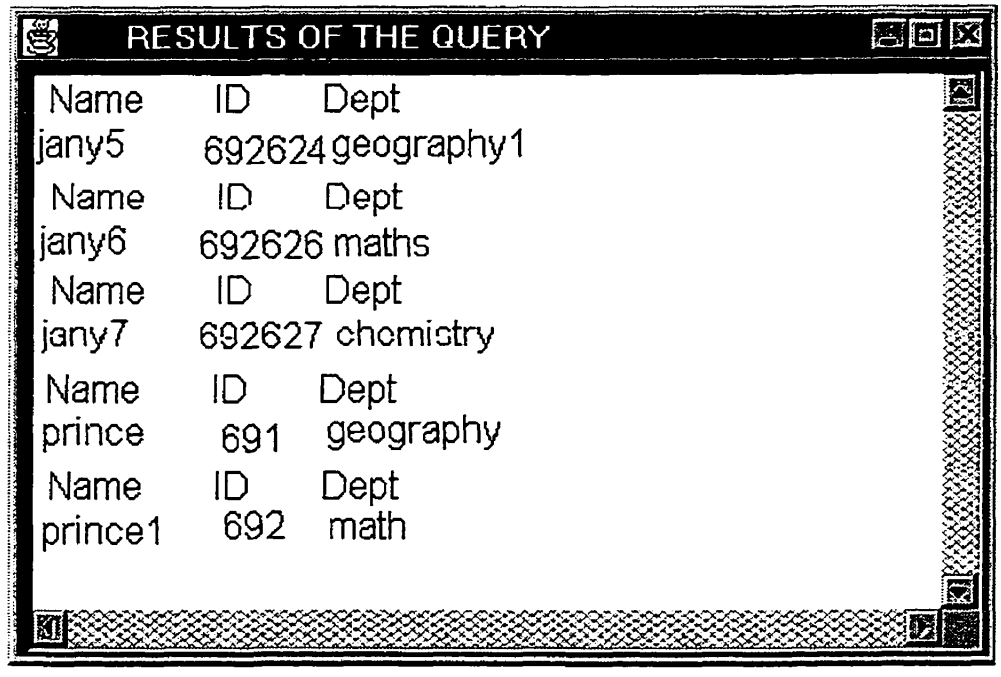

Figure 4.7 Information Displayed to User with Security Level 3

Figure 4.6 shows the results obtained by executing the query sent by user with security level 1. Now lets us consider the information obtained by the user with security level 3.

Figure 4.7 illustrates the information that is displayed to a user with security level-3 clearance.

Figure 4.8 displays the information that can be used by the database administrator (human guard) for audit purposes. The information in Figure 4.8 shows how the SSA prototype processes the information.

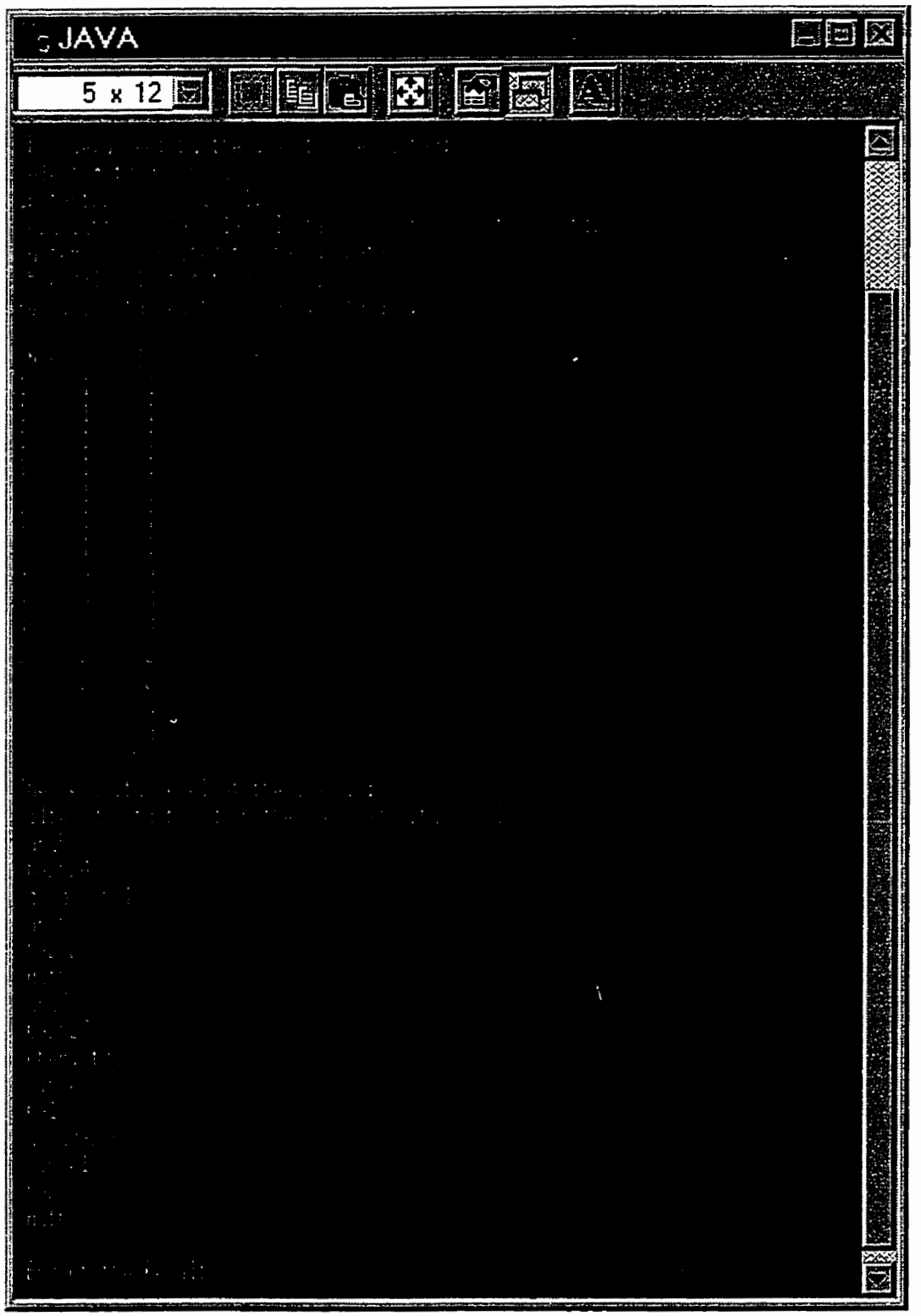

Figure 4.8 Displays the information that will be stored in the Audit Record.

4.5 Performance Comparison

Generally speaking, database performance depends on a lot of factors, including system configuration, security needs, database design and others. The smallest variations in a system configuration can radically alter the results. The only way to determine conclusively how applications will fair under any database is to run queries.

In our case we have considered the performance characteristics between the instantDB system before implementing SSA system and after implementing SSA system. To develop a performance chart, the access speed is used as the benchmark. Tables with 20000, 40000, 60000, 80000 records were considered and we accessed them using a simple user query. The average speed on 10 different iterations was taken for each set of (i.e. 20000 - 10 iterations, 40000 - 10 iterations etc.) records. Each table has five columns Name, Userid, Department, year_of_study, and gender, with the associate's data type as follows: varchar(40), Userid int, Department char(30), year_of_study int, and gender char(1). The query used for the test purpose is:

SELECT * FROM TABLE_NAME

Figure 4.9 shows the time taken to access x number of records. Notice from the figure that the access time taken by the SSA is higher than the access time of the InstantDB. There are two main reasons for this kind of behavior. First, the SSA accesses two databases while the instantDB accesses only one. Second, after accessing the database and data catalog the SSA has to perform data filtration based on the user security level. Other than these two reasons there are some minor overheads, like modification of the query to retrieve information from the data catalog and logical mapping of the security level information in the data catalog. Recall, in Section 2.4.4 that this thesis

stresses more on the importance of security in the MDBS than on the efficiency and accessing speed of the database.

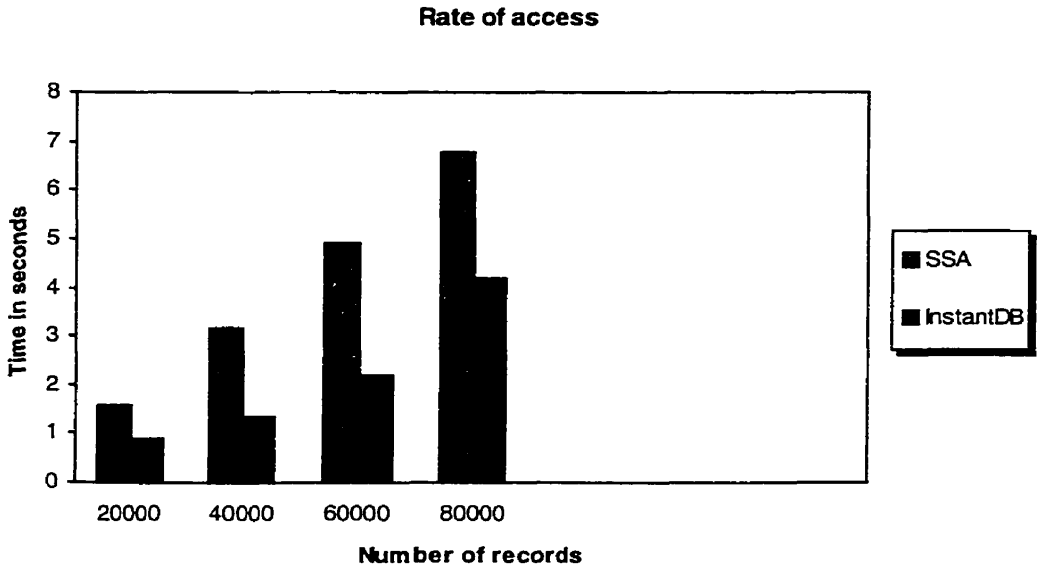

Figure 4.9 Comparison between SSA and InstantDB.

Here are some suggestions to improve the efficiency and the accessing speed of the SSA system. Some of them are:

- Find a new approach to perform logical mapping between the data in the database and security levels in the data catalog.
- Using efficient database structures, such as HEAP and HASH instead of B-TREE. Topology analysis during loading can trigger bypass of unnecessary run-time action [8].
- Using Optimizers, such as SQL-Optimizer is a software product that automatically analyzes and optimizes SQL statements. SQL-optimizer monitors live databases and identifies slow and bad (not optimized) running SQL transactions (Select, Insert, Update, Delete) interactively. When detected, the SQL-Optimizer recommends an optimized reformulated SQL statement to replace the original statement received by the database server.

Chapter 5

Conclusion and Future Work

In a multidatabase system security management is a major issue because it is very difficult to have a security system that is transparent, secure, easy to manage, and efficient. There is always a trade-off between security and the efficiency of the database system.

This thesis introduces a Secure Software Architecture (SSA) design, which is developed based on the evaluation criteria of U.S Department of Defense for a secure system [11]. It also provides a new approach to solve the cascade problem using a prototype multilevel security model. In this thesis a new design for multi-level secure software architecture is proposed. An experimental prototype of the design is developed and functionally tested. The preliminary performance results are presented in Section 4.5.

The implementation of the prototype shows that it is useful to have security components, which provides mechanisms to implement multilevel security and data filtration. These mechanisms help in isolating users at different security levels and authenticating them before sending the query to the database. This prevents an untrusted user from disguising themselves as a user at another security level. Recall, cascade problem from Section 2.3 where we described the problem and how it effects the multi-level security system. Many efficient algorithms have been proposed in the literature to deal with the cascade vulnerability detection. However, there is not much research that deals with designing and implementation of secure software architecture to solve the cascade problem. This thesis provided a new design and implementation

of security components that prevents users from changing the security levels of the data or degrading the information.

Let us consider an example and see how cascade problem is avoided by the SSA, A user ***"JOHN"*** with security level Top-Secret would like to share information with a user ***"SAMUEL"*** with security level Non-Confidential. The information is stored in a DMBS system where the multi-level security is provided by attaching the security level of the data to the data itself (physically attached), as shown in Table 5.1.

Table 5.1 -- School Table

Name	Security_Level	ID	Security_Level
John	Top-Secret	6495527	Top-Secret
Mary	Secret	9697970	Secret
Funky	Confidential	8656969	Confidential
Samuel	Non-Confidential	5866969	Non-Confidential

"JOHN" can degrade the information in the School Table. So ***"SAMUEL"*** can read the information in that table when he performs a query. This is possible since ***"JOHN"*** knows the security level of the data that is to be changed. So ***"JOHN"*** changes the security level of the data and tells ***"SAMUEL"*** to access it. In this case there are no security components to prevent ***"JOHN"*** from changing the security levels. Because of this the security is compromised in the DMBS system since there are no security components to prevent the security levels from being changed and the security levels are also visible to the user. But in case of SSA, the security levels are not attached to the data (i.e., only logical link exists between the security level and the data) and they are not attached to the data. So ***"JOHN"*** does not know what the security levels of the data are, Hence, he cannot degrade the security level of the data. Thus, the security components in the SSA prevent the users from changing or degrading the security levels of the data. This approach can also be used where the

data is distributed across different databases. So the SSA provides a new approach to solve the cascade problem by providing solution using a secure architecture design rather than using efficient algorithms to solve the cascade problem. The remainder of this chapter focuses on the performance, general SSA approach, future research, and conclusion.

5.1 General SSA Approach And Conclusion

The SSA system provides the required security based on the concept of multi-level security and transparency. This is achieved by partitioning the data into two databases. The first database (LDBS) contains the data (i.e. elements in the table) and the second database (LDC) consists of security levels (i.e. security level in the table). These two databases are logically linked to one another by the security components described in Section 3.1 and 3.2. When users send queries to the SSA system, security components in the SSA establish a logical link between the databases and get the security level of the user. Once the results from both the databases are retrieved, the security component filters the results based on the security level of the user. The filtered data is then displayed to the user.

The key features of the SSA approach are (1) high level of security since it provides a new approach to solve the cascade problem and (2) designed based on the U.S Department of Defense evaluation criteria for a secure system.

The SSA followed the three main categories defined by the U.S. Department of Defense in Section 2.3 for development purposes. In case of *security policy* SSA follows both DAC and MAC policies. This is done with the help of the security administrator, which provided user authentication and verification. Data catalogs provided multi-level security and data filtration. *Accountability* issues are taken care

of with the help of audit records and user identification (security administrator). The *assurance* category is beyond the scope of this thesis since it deals with the hardware/software/firmware components, but a effort was made to make sure that there were no software conflicts between the SSA and the operating system.

Other features of the SSA approach are the transparency to the user, an acceptable access time when considered in the context of the high level of security it provides, and the architecture can be easily adapted on top of any distributed or multidatabase systems.

There are certain limitations in that the SSA system is slow when we compare it with other distributed systems, additional software has to installed at the local sites, the data format plays an important role in the SSA system, and testing of the system has to be performed more thoroughly.

5.2 Future Research

This thesis contributes a secure software architecture design in multidatabase systems. The thesis has introduced a new type of security architecture for the MDBS, targeted towards multilevel-security, transparency, and data partitioning concept.

The SSA system was compared with other security approaches (recall Section 2.6). Run-time performance analysis was performed with respect to the InstantDB system before and after implementing the SSA design, improvements to the performance of the SSA system are suggested.

This thesis assumed that the environment is homogeneous. That is, the multi-level security and the databases are designed and operate identically. A more realistic

environment would have some form of heterogeneity. This is one of the future research directions where considerable amount of work is requires. In this thesis, we have implemented only the LDBS and analyzed its performance. The implementation and analysis of the complete architecture and its performance are other areas where future work can be done. As part of future work, an investigation of the impact of the system environment modifications and other system tuning suggestions mentioned in Section 4.5 on the performance of the SSA system is necessary.

Reference

[1] Atluri V., Jajodia S., and Bertino E., "Transaction Processing in Multilevel Secure data bases with Kernelized Architecture: Challenges and Solutions". *In IEEE Transactions on Knowledeg and Data Engineering, Vol 9, No. 5, September/October 1997, page 697-708.*

[2] Barker, K., "Quantification of Autonomy on Multidatabase Systems", *The Journal of System Integration*, Kluwer Academic Publishers, *1993, pages 1-26.*

[3] Barker, K., M. Evans and J. Anderson, "Measuring Autonomy in Heterogeneous Cooperative Systems", *AAAI Workshop on Cooperation Among Heterogeneous Intelligent Systems, San Jose, California, U.S.A., July 1992, pages 11-16.*

[4] Bertino E., Jajodia S., Mancini L., and Ray I., "Advanced Transaction Processing in Multilevel Secure File Stores". *IEEE Transactions on Knowledge and Data Engineering, Vol. 10, No.1, January/Feburary 1998, page 120-135.*

[5] Bright M., "Security Issues and Metadata Requirements in Multidatabases", *Colloquium at University of Pittsburgh, November 1997.*

[6] Bauer M., Coburn N., Larson P.A., and Martin P., "Managing global information in the CORDS Multidatabase System". *In Proceedings of 2nd International Conference on Cooperative Information Systems (CoopIS'94), Toronto, Canada, May 1994.*

[7] Breitbart Y., Garcia-Molina H., and Silberschatx A., "Transaction Management in Multidatabase Systems", *Modern Database System, page 573-591, 1995.*

[8] Comer D., "The Ubiquitous B-Tree", *ACM Computing Surveys Volume 11, number 2, June 1979.*

[9] Courtney R.H., "Factors affecting the availability of security measures in data processing system components". *Proceedings of 13th National Computer Security Conference. October 1990, Page 54-66.*

[10] Castano S., Fugini M., Martella G., and Samarati P., *Database Security. Addison-Wesley Publishing Company, 1994.*

[11] Castano S., *Database Security, Addison –Wesley publications, 1995.*

[12] Coexistence of Relations and Objects in Distributed Object Computing, "IBM DataJoiner",http://www.acl.lanl.gov/sunrise/dbms/datajoin.html, *1998.*

[13] Chrysanthis P., and Ramamritham K.," Autonomy Requirements in Heterogeneous Distributed Database Systems", *Proceedings of the 6th International Conference on Management of Data (COMAD'94), December 1994.*

[14] Deen, S.M., Amin, R.R., et al., "The Architecture of a generalized Distributed Database System - PRECI*", *The Computer Journal, pp. 282-290, vol. 28, no. 3, 1985.*

[15] Department of Defense Computer Security Center. Department of Defense Trusted Computer System Evaluation Criteria, *December 1985.* http://nsi.org/Library/Compsec/sec0.html

[16] Deng R.H., Shailendra K.B., Wang W., and Aurel A.L., "Integrating Security in CORBA Based Object Architectures", *Theory and Practice of Object Systems 3(1), 1997.*

[17] Dean D., Felten E.W., Wallach D.S., "Java security: From HotJava to Netscape and Beyond". In *Proceedings of 1996 IEEE Symposium on Security and Privacy, Oakland, California, May 1996.*

[18] Defense Information Systems Agency/Joint Interoperability. *Department of Defense, USA.,* http://nsi.org/ *December 1985*

[19] Essmayr W., Kastner F., Pernul G., and Tjoa A.M., "The Security Architecture of IRO-DB", *In Processing of 12th IFIP International Conference on Information Security, Island of Samos, Greece, May 1996.*

[20] Fernandez E.B., Summers R.C., and Wood C., *Database Security and Integrity. Reading, Mass.: Addison-Wesley, 1981.*

[21] Georgakopoulos D., Rusinkiewicz M., and Sheth A., "Using Tickets to Enforce the Serializability of Multidatabase Transactions", In *IEEE Transactions on Knowledge and Data Engineering, December 1993.*

[22] Gardarin G., Gannouni S., Finance B., Fankhauser P., Klas W., Pastre D., Legoff R., Ramfos A., "IRO-DB: A Distributed System Federating Object and Relational Databases". *Object-Oriented Multidatabase Systems, Prentice Hall.*

[23] Horton J.D., Harland R., Ashby E., Cooper R.H., Hyslop W.F., Nickerson B.G., Stewart W.M. and Ward O.K., "The Cascade Vulnerability Problem". In *1993 IEEE Computer Society Symposium on Research in Security and Privacy, September 1993.*

[24] Ilgun K., "USTAT: A Real-time Intrusion Detection System for UNIX". In *1993 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland California, May 1993, pages 16-28.*

[25] Jajodia S. and Sandhu R., "Toward a Multilevel Relational Data Model". *Proceedings of ACM-SIGMOD International Conference on Management of Data. Denver, Colorado, May 1991, pages 50-59.*

[26] Lindqvist U., Johnsson E., "How to systematically classify computer security intrusions", *In 1997 IEEE Computer Society Symposium on Security and Privacy, Oakland California, May 1997, pages 154-163.*

[27] Lunt T.F., "A Survey of intrusion detection techniques", *Computers & Security, Elsevier Sc. (North-Holland), 1993.*

[28] Martin P. and Powley W., "Storing MULTIDATABASE SYSTEM Catalog information in an X.500 Directory Service". *In Proceedings of the 1994 CAS Conference, Toronto, Canada, October 1994, page 216-226.*

[29] McDermott J.P., Jajodia S., and Sandhu R.S., "A Single-level Scheduler for the Replicated Architecture for Multilevel-secure Databases". *In Proceedings of Seventh Annual Computer Security Applications Conference, San Antonio, Texas, December 1991, pages 2-11.*

[30] Marjorie T., David B., et al., "Mermaid--A Front-End to Distributed Heterogeneous Databases", *Procceding IEEE, pp. 695-708,vol 75, no. 5, May 1987.*

[31] Moriconi M., Qian X., Riemenschneider R.A., and Gong L., "Secure software Architectures". *In 1997 IEEE Computer Society Symposium on Security and Privacy, Oakland California, May 1997.*

[32] Özsu M.T., and Valduriez P., *Principles of Distributed Database Systems. Prentice Hall, Englewood Cliffs, New Jersey, 1994.*

[33] Pesati V.R., Keefe T.F., and Shankar, "The design and implementation of a multilevel secure log manager", *In 1997 IEEE Computer Society Symposium on Security and Privacy, Oakland California, May 1997, pages 53-64.*

[34] Russel D. and Gangemi G.T., *Computer Security Basics. O'Reill & Associates Inc., 1991.*

[35] Sandhu R.S. and Jajodia S., "Integrity mechanisms in database management systems". *Proceedings of NIST-NCSC National Computer Security Conference, Washington, D.C., October 1990, pages 526-540.*

[36] Shekhar S., *Database Security: An Introduction. UMI Research Press 1995.*

[37] Schuba C.L., Krsul I.V., Kuhn M.G., Spafford E.H., Sundaram A., and Zamboni D., "Analysis of a Denial of Service Attack on TCP", *In 1997 IEEE Computer Society Symposium on Security and Privacy, Oakland California, May 1997, pages 208-223.*

[38] Sheth, A. and Larson, J., "Federated Database Systems for Managing Distributed , Heterogeneous, and Autonomous Databases", *In ACM Computing Surveys, Vol. 22, No. 3, September 1990, P183-236.*

[39] Saake G., Conrad S., Schmitt I., and Türker C. "Object-Oriented Database Design: What is the Difference with Relational Database Design", *ObjectWorld Frankfurt'95, 1995.*

[40] Sandhu R.S., "On Some Research Issues in Multilevel Database Security", *Proceedings of 5th RADC Workshop on Multilevel Database Security, Fredonia, PA, October 1992.*

[41] Thuraisingham B., and Ford W., "Security Constraint Processing in a Multilevel Secure Distributed Database Management System". *In IEEE Transactions on Knowledge and Data Engineering. Vol. 7, No.2, April 1995.*

[42] Vishu K., Su Y. W., et al., "A Distributed Database Architecture for an Integrated Manufacturing Facility", *In Second Symp. Knowledge-Based Integrated Info. Sys. Eng., May 1987.*

[43] Qian X. and Lunt T.F., "A Semantic Framework of the Multilevel Secure Relational Model". *In IEEE Transactions on Knowledge and Data Engineering, Vol. 9, No. 2, March/April 1997, page 292 -301.*