2005

Area access control systems: Zone management and personnel tracking

Bharath Natarajan

University of South Florida

Follow this and additional works at: http://scholarcommons.usf.edu/etd

Part of the American Studies Commons

Scholar Commons Citation

Natarajan, Bharath, "Area access control systems: Zone management and personnel tracking" (2005). Graduate Theses and Dissertations.
http://scholarcommons.usf.edu/etd/787

This Thesis is brought to you for free and open access by the Graduate School at Scholar Commons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact scholarcommons@usf.edu.
Area Access Control Systems:
Zone Management And Personnel Tracking

by

Bharath Natarajan

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Engineering Management
Department of Industrial and Management Systems Engineering
College of Engineering
University of South Florida

Major Professor: Ali Yalcin, Ph.D.
Michael Weng, Ph.D.
José Zayas-Castro, Ph.D.

Date of Approval:
July 12, 2005

Keywords: objects, recursive querying, recursive relationships, object-oriented database
management systems, unary relationships

© Copyright 2005, Bharath Natarajan
DEDICATION

To My Beloved Parents,
Dr. M. Natarajan and Thuriya Natarajan
# TABLE OF CONTENTS

LIST OF TABLES iii

LIST OF FIGURES iv

ABSTRACT v

CHAPTER ONE: INTRODUCTION AND MOTIVATION 1

1.1 Area Access Control Systems 1
   1.1.1 Working Principle of an Area Access Control System 2
   1.1.2 General Access Control Systems Requirements 4
   1.1.3 Multisensor Fusion and Biometrics 6
1.2 Area Access Control Database System 7
1.3 A Centralized Area Access Control System 10
1.4 Motivation 12

CHAPTER TWO: PROBLEM DESCRIPTION, PROPOSED SOLUTION AND RELATED LITERATURE 15

2.1 Problem Description and Objective 15
2.2 Proposed Data Model 15
   2.2.1 A Relational Database Solution 16
   2.2.2 Relational Database's Limitations in Recursive Relationships 18
2.3 Recursive Querying 20
   2.3.1 Recursive Procedures and Functions 22
2.4 Object oriented Databases – An Overview 23
2.5 An Object Oriented Database Solution to Recursive Querying 25
2.6 An Object Oriented Implementation 27

CHAPTER THREE: OBJECT ORIENTED AREA ACCESS CONTROL DATABASE SYSTEM IMPLEMENTATION 30

3.1 Data Model and Schema Generation 30
3.2 Zone, Person and Transaction Class Definitions 30
   3.2.1 Zone Class 32
   3.2.2 Person Class 32
   3.2.3 Transaction Class 32
3.3 Database Setup and Population 33
3.4 C++ Implementation 33
   3.4.1 Zone Management 34
   3.4.2 Personnel Tracking 35

CHAPTER FOUR: RESULTS 38
   4.1 Zone Management 39
   4.2 Personnel Tracking 40

CHAPTER FIVE: CONCLUSIONS, WORKING SCHEME AND FUTURE WORK 47
   5.1 Conclusions 47
   5.2 Working Scheme of the Object Oriented Area Access Control Database System 49
   5.3 Future Work 52

REFERENCES 53

APPENDICES 56

   Appendix A: C++ Program for Object Oriented Area Access Control Database System 56
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1:</td>
<td>Table Corresponding to the Persons Authorized to Enter Zone 3</td>
<td>9</td>
</tr>
<tr>
<td>Table 2:</td>
<td>Table Corresponding to the Transactions through Access Point (C)</td>
<td>9</td>
</tr>
<tr>
<td>Table 3:</td>
<td>BOM Parts Table</td>
<td>20</td>
</tr>
<tr>
<td>Table 4:</td>
<td>BOM Cross Reference Table</td>
<td>20</td>
</tr>
<tr>
<td>Table 5:</td>
<td>Records Corresponding to the Location of Person (987-65-4321) at 10:00:00</td>
<td>42</td>
</tr>
<tr>
<td>Table 6:</td>
<td>Records Corresponding to the Location of Person (123456789) between 9:00:00 and 11:00:00</td>
<td>44</td>
</tr>
<tr>
<td>Table 7:</td>
<td>Records Corresponding to the Personnel Interaction between (123-45-6789) between 9:00:00 and 11:30:00</td>
<td>46</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Block Diagram for Verification and Identification Operating Modes</td>
<td>3</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Hypothetical Area Access Control Case</td>
<td>8</td>
</tr>
<tr>
<td>Figure 1.3:</td>
<td>Using Two Tables for Entire Installation</td>
<td>11</td>
</tr>
<tr>
<td>Figure 1.4</td>
<td>Centralized versus. Non-Centralized Database Implementation</td>
<td>12</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>ER Diagram of a Relational Access Control Database System</td>
<td>17</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>Graph Representation of BOM Recursive Data Structure</td>
<td>19</td>
</tr>
<tr>
<td>Figure 2.3:</td>
<td>Directed Graph Representation of the Zone Objects in the Data Structure</td>
<td>26</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>UML Model for an Object Oriented Area Access Control Database System</td>
<td>28</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>ODL File for Area Access Control Systems’s Object Oriented Database</td>
<td>31</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Hypothetical Area Access Control System</td>
<td>39</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Program Output for Zone Management</td>
<td>40</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Program Output for Tracking: Location of Person at Specific Time</td>
<td>41</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>Program Output for Tracking: Location of Person between Two Time Limits</td>
<td>43</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>Program Output for Tracking: Personnel Interaction between Two Time Limits</td>
<td>45</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Working Scheme of the Centralized Object Oriented Area Access</td>
<td>50</td>
</tr>
</tbody>
</table>
AREA ACCESS CONTROL SYSTEMS:
ZONE MANAGEMENT AND PERSONNEL TRACKING

Bharath Natarajan

ABSTRACT

Area access control is defined as the process of mediating requests to enter a physical area through one or more entry points. Area access control database systems are the collections of information required for an access control system to access, query, retrieve and match real time user inputs with persistent data to ensure the integrity of the resources it protects. This thesis presents an object oriented approach to the design and implementation of a centralized area access control database system and focuses on two features, zone management and personnel tracking. Zone management is defined as the process of hierarchically relating a zone to other immediately adjacent zone(s) that a user is required to have prior access to. This feature will automatically generate all zones that a user requires prior access to in order to approach a target zone. To implement zone management, the database system is required to support recursive relationships and recursive querying. The personnel-tracking feature allows the administrator to obtain information such as the movement of persons of interest and their interactions with others in the installation at any particular time. The results of this thesis contribute to the implementation of a sophisticated area access control database system capable of handling multiple installations, and generating the access rules and paths for each new
user automatically. In addition, the object oriented area access control database system is able to support unconventional data types such as images and sound which are essential for emerging biometric security systems.
1. INTRODUCTION AND MOTIVATION

Area access control is defined as the process of mediating requests to enter a physical area through one or more entry points that are maintained by a system which determines whether the request should be granted or denied. The requests are often made by providing user’s identification information which is used by the access control system to verify if that user has the required privilege to access the requested location. Such systems use databases to store information required for the process of verifying authenticity of users. These databases are called area access control database systems. They must provide accurate and consistent service to the access control system to ensure that legitimate users are not denied access and illegitimate users are never entertained.

This research examines the functionality and working of an area access control system, its database requirements, and the advantages and disadvantages of the various types of database implementations. The results of this thesis contribute to the design of a sophisticated area access control database system and its implementation.

1.1 Area Access Control Systems

Area access control systems control access into an area and ensure that only authorized access can take place. Area access control systems are merely hardware extensions to software database access control systems since the area access control system ultimately accesses a database of authorized or unauthorized entities. This is done
by utilizing system administrator specified access control policies that should be checked and satisfied in order to access the managed areas. There are several applications for area access control systems. These systems are used to regulate entry or exit of people into rooms, laboratories, airports, prisons and other secure installations.

1.1.1 Working Principle of an Access Control System

This section discusses the general principles of an area access control system. Area access control is based on a user identifier (UID), a unique identification value, and possibly a group identifier (GID) which each user is assigned [1]. A user approaches the entrance to an area into which he/she wishes to enter and identifies him/herself to the access control device. There are two modes in which an access control system operates. They are the verification mode or identification mode [2].

In the verification mode the user claims his identity using an access card or other means and then verifies this information with a PIN (person identification number) or biometric input. In systems that use this mode of operation, the system conducts a one to one comparison to determine if the claim is true. In the identification mode, the user directly enters the PIN or biometric input and the system searches the database for a match. In this case the system conducts a one-to-many comparison to verify user’s claim. Figure 1.1 shows the block diagrams showing how each mode functions.
When a user operates the user interface and the user is authenticated, an access token is created and it is associated with every access control process executing on behalf of the user [1]. An access token is a set of values containing the system privileges of the user and of the groups in which the user is a member. This information is obtained from the UID and GID domains of the user’s record. The access token provides the information that is used to compare with the user’s privilege request with the user’s authorized privileges stored in the database. The current access control system databases are simple relational databases which contain tables of domains that are matched with the values of the access tokens to verify if the user has the required privileges [1].
1.1.2 General Access Control Systems Requirements

An area access control system much like other access control systems must support some general access control system requirements, concepts and principles [3]:

1.1.2.1 Reliable Input. This concept requires that the access control must obtain proper input which cannot be faked. An example is the use of an IP address to check authenticity which is not a characteristic of the user but the terminal and such an input may be masked or duplicated by an illicit user. The prerequisite for access control should be the requirement of proper user authentication [4].

1.1.2.2 Fine or Course Access Rules and Specifications. The policies and access rules should be chosen so that they are not too fine nor too coarse. To illustrate this consider a specification of access rules for every single user when the user is a member of a group of users or objects that have the same access rules. It would ease the burden on access control administration to share access rules and in that case, a single access control rule should be shared and specified for the group of users, group of objects and possibly even a group of actions [5]. In some organizations, the access control rules may be naturally associated with their organizational roles [6]. An example of such an organization is the military where the roles of an armory officer and an information specialist officer will require different authorization rules to enter their respective work environments.

1.1.2.3 Least Privilege. This principle states that, typically, when access is granted to a user, the user only obtains the least privilege required to perform his task. This is a rather significant concept. If the concept of privileges were introduced for users in addition to individual access specifications, the access control device can first check
for the least privilege needed to access the area. This avoids unnecessary specifications for low security areas such as the outer entrance that every user has the privilege to enter.

1.1.2.4 Separation of duty. This is a principle that states that no user should be given enough privileges to misuse the system on their own [7]. To obtain the privileges required to avoid this situation, the historical based information is used to assess the privilege needed.

1.1.2.5 Open Class (Negative Identification) and Closed Class (Positive Identification). The discretionary policies may be distinguished into two classes as open or as closed [3]. In the case of the closed class or positive identification rules, the access control database contains the authorized user listing and the controller checks to see if each user is listed or meets the require privilege. In the case of open class or negative identification, the users are assumed to have access they are check to see if they are specific unauthorized or banned users. The open class rules maybe used in areas with public access such as airports to identify known terrorists or in prisons to identify if the user requesting exit from an area is a convict.

1.1.2.6 Administrative Policies. Finally, perhaps the most significant rule in area access control systems is the need to specify and define the administrative policies that determine and regulate who is allowed to add, delete, or modify the access privileges of a user. The administrative policies may be of four types; centralized, where a privileged user or group reserved the privilege of granting or revoking all authorizations in the company; hierarchical/cooperative, where a set of authorized users grant or revoke privileges; ownership, where each user or object is associated
with an owner or supervisor who decides the privileges; and decentralized, where the
owner or administrator delegates other users the privilege or specifying authorizations
[1].

1.1.3 Multisensor Fusion and Biometrics

Most access control devices are quite reliable however they are not fool proof and they all have limitations that may cause an access control system to malfunction. In such cases, the accuracy of an access control system may be enhanced by up to 60% if two or more devices are fused and used together at an entry point. The use of multiple devices or sensors in this manner is termed multisensor fusion [8].

Biometric recognition or simply biometrics refers to the automatic recognition of individuals based on their physiological and/or behavioral characteristics [2]. The use of biometrics in access control systems is becoming more and more common and the trend has followed into the area access control scenario. The advantage of this type of identification method is the possibility of confirming the identity of individuals based on their characteristics rather than what they possess. Cards, keys and other material identifiers are easily lost or reproduced.

A question arises on the relevance of multisensor fusion and biometric systems on the area access control database systems. The relevance is in the need for current area access control database systems to keep up with the current and future trend of access control systems. The database systems supporting these access control systems must support biometric data types, namely, patterns, images, and sounds [9,10]. The database
systems must also allow the use of multiple devices at access points without a compromising overhead.

1.2 Area Access Control Database System

Traditionally, databases are defined as collections of textual or numeric information organized in a form that enables a computer program to store, access and retrieve persistent data quickly and accurately [11,12]. Access control databases are the collections of information required for an access control program to access, query, retrieve and match real time user inputs with persistent data accurately in order to ensure the integrity of the resources that it protects.

To motivate the objectives of this research, a simple illustrative case is considered. Figure 1.2 depicts the layout of a hypothetical installation with multiple areas of various security levels. It consists of five zones that require specific access through designated access points.
In the implementations described in literature on access control systems, each access point to a new zone consists of one or more independent access control device(s) such as a keypad, swipe-card reader or fingerprint scanner. Each of these devices is a standalone system referencing a data table consisting of records of authorized users. The user entry is matched with the record entries in the data file to determine authorization privileges. Table 1 and Table 2 illustrate the two tables that would be required at the access point (C) into Zone 3 in such a database.
This access control database system models an environment consisting of an authorized access table and a transactions table. The authorized access table will contain records of each employee in the organization with legal authorization into the zone through that access point and a transaction table contains records corresponding to each transaction through an access point and their pertaining information. Most access control systems will function in a similar manner with a table of authorized access and a transaction table for each access point.
Area access control systems, sensors and devices are becoming more accurate, and the identification technology and algorithms are improving significantly. However, such remarkable improvements have not been reported in the database aspects of access control systems. There has been no significant literature on databases specifically designed for addressing the needs of the access control systems. In publications as recent as 2002, the database for access control systems is mentioned as merely a table [1].

1.3 A Centralized Area Access Control Database System

This thesis describes the use of a centralized database for single installation. If current database implementations are utilized, several individual tables are required corresponding to each access point. In many cases these tables will contain duplicate records also present in other tables i.e. the table for the access point into zone 4 will contain records that are also present in the tables for access points into zones 1, 2 and 3.

The use of a centralized database system reduces the overhead due to the existence of duplicate records of a person in multiple tables. To illustrate this, consider the use of two tables, a person table and a zone table, for the entire installation as shown in Figure 1.3. For the enrollment of a new user, the addition of a new record in the person table is sufficient, versus the need for a record in multiple tables at each entry point. By building relationships (accesses_zone) between each person record and the record of the zone that the person requires access to, the task of enrollment is accomplished.
The centralized database requires the use of only one new relationship rather than multiple relationships for each enrollment. Such a relationship is established between the new person record of the person table and the record corresponding to the final target zone requiring access in the zone table. In addition, relationships are established between the records of the zone record and all the immediately adjacent zone record(s) that need to be accessed in order to get to the target zone. The relationship specification between the zones is a one time administrative process that is done when the database system is initialized and it is specific to the installation. This feature is called zone management and it requires only one new record and one additional relationship, to the target zone, each time a new person is enrolled to the system.

Zone management is defined as the process of hierarchically relating a zone to other immediately adjacent zone(s) that a user is required to have prior access to, in order to enter the target zone.
In the example shown in Figure 1.2, consider a person requiring access to zone 4. A single relationship (accesses_zone) is built between a person entity and the zone entity corresponding to zone 4. The pre-existing relationships between the zones will ensure that the person has access to all preliminary zones that need to be accessed. This is diagrammatically shown in Figure 1.4.

1.4 Motivation

A centralized access control database system has the advantage of allowing a single database system to operate all access points, and the implementation of the zone management feature. From Figure 1.4, one can intuitively infer that the process of setting up and administering the database is considerably simplified in a centralized system equipped with zone management.
Relational Database Management Systems (RDBMS) are very successful in representing many traditional applications such as order processing and inventory control. However, RDBMSs have been found to be inadequate in engineering systems with large volumes of data with simple data structures or few instances of data with complex data structures [11,13,14].

The availability of reliable biometric devices combined with decrease in costs of biometric systems has brought about a new trend in access control systems in general. The use of these biometric systems in area access control systems have increased over the last few years and this trend will likely continue in the coming years. The use of these systems will bring about the need for databases that support unconventional data types such as images and sound [2]. The challenge of unconventional data types was earlier encountered when database support was required for CAD/CAM systems. RDBMSs were found to have difficulties addressing this challenge.

Object oriented database management systems (OODBMS) were found to adequately address the need for unconventional data types in CAD/CAM database systems [11]. Further, object oriented databases are able to support recursive data, establish unary relationships and perform recursive queries [11,13,14]. By recursively relating the objects of the zone class, zone management can be introduced.

Relationships in which entities are related to one or more other entities of the same table as described above are called recursive or unary relationships [11,14]. Implementing recursive relationships in traditional database systems is challenging as explained in greater detail in Section 2.2.2.
The RDBMS’ inadequacies and limitations described in this section are precisely the characteristics that are essential for the area access control database system. Thus it is critical that alternate database management systems are explored for the area access control database application.
2. PROBLEM DESCRIPTION AND PROPOSED SOLUTION

2.1 Problem Description and Objective

This thesis aims to design and implement a single centralized area access control database system that supports zone management. To design such a system, an appropriate data model must be selected. Database systems and packages that support the desired data model need to be identified and an implementation methodology must be determined. Support for recursive relationships and recursive querying are the primary requirements that need to be addressed by centralized area access control database systems in order to implement zone management. This research also aims to provide a personnel tracking feature which allows the administrator to obtain information such as the movement of suspicious persons and their interactions with other persons in the installation at any particular time. Personnel-tracking is accomplished by analyzing the transaction entries and the privilege levels of the suspicious person and other persons within the facility.

2.2 Proposed Data Model

This section describes an area access control database system model that addresses the objectives and the challenges that were described in the earlier sections.
2.2.1 A Relational Database Solution

The relational database management systems (RDBMS) are databases which are organized as sets of formally-described tables, fields, records, and files. A field is a single piece of information; a record is one complete set of fields; and a file is a collection of records. Several RDBMS packages exist today, such as MS Access and Oracle.

In the relational model, data is viewed conceptually in tables which are defined as the logical view of related data that capture the concepts of entities and attributes [12]. An entity is the representation of a basic object with a physical existence such as a person, a car, an employee and a student; or with a conceptual existence such as transactions. Every such entity has attributes that describe the entity. Each attribute of an entity will have a value associated with it. The final relational concept that needs to be defined is a relationship which is a description of the association between two or more entities.

The Structured Query Language (SQL) has been defined by the relational database community to manipulate and query data in one or more tables. This language consists of a set of basic keywords, their meaning and syntax. There are several versions of SQL used by various RDMS packages either by extending or limiting the keywords that may be used. However, the American National Standards Institute (ANSI) SQL is considered the standardized SQL. The standard SQL set contains about 30 basic instructions.

In the introductory chapter of this thesis, current area access control database systems were described to be inefficient and at times redundant with duplicate records in multiple tables. An alternative approach is the use of a single, centralized, three-table
database for the entire installation, one table for employees, another for zones and a final table for the database transactions. By building relationships between employees and zones data redundancy can be avoided. Furthermore, zone management can be established by incorporating recursive relationships between the zones. An ER diagram of such a system is shown in Figure 2.1 to illustrate an implementation of an access control system with a recursive relationship between the zones.

![ER Diagram of a Relational Access Control Database System](image)

**Figure 2.1: ER Diagram of a Relational Access Control Database System**

In the ER diagram shown in Figure 2.1, the zone table consists of records corresponding to each zone in the physical installation and their pertaining information. These records each participate in a recursive relationship, `shares_access_point`, with one or more other records of the zone table. The recursive relationship `shares_access_point`
relates the target zone to a zone that either encloses the target zone, or is adjacent to it, and needs to be accessed to approach the target zone. Hence a hierarchy is established between the outer zones and the inner target zones. The personnel table consists of records corresponding to each person who requires access to any zone within the installation and their attributes. Personnel records also possess the unique attributes corresponding to different access control devices such as the swipe-card number, fingerprint code, iris image code etc. that are matched at the access points. An additional table that is required is the transactions table. The transactions table consists of records which contain information on each transaction in the access control database system. The transaction record contains a timestamp, person id and zone access data, which allows analysis, monitoring and tracking of personnel.

2.2.2 Relational Database’s Limitations in Recursive Relationships

Recursion is characterized by the presence of records in a single table that are related to each other via primary-key or foreign-key relationships [15]. Relationships built in this way are termed recursive relationships or unary relationships [11]. Though the actual ER diagram shown in Figure 2.3 is a feasible relational database design, implementation of such a system would come with considerable overhead due to the need for cross-reference tables [15]. An example of an application for recursive relationships is the BOM problem that has been addressed using relational databases. Consider a BOM example consisting of five parts 1, A, B, c and d with an assembly structure as shown in the Figure 2.2.
Figure 2.2: Graph Representation of BOM Recursive Data Structure

Figure 2.3 shows the parts table and the cross-reference table corresponding to the recursive data structure shown in graph representation of the BOM recursive data structure, shown in Figure 2.2. It is seen that in order to relate records of the parts table to other records of the part table through a relationship composed_of, a new cross reference table is required containing all possible relationships. It is seen in the Cross Reference Table shown in Figure 2.3 that five records are required corresponding to part 1 even though part 1 only participates in two direct relationships. This is because even though parts c and d are not direct children of part 1, they are in fact subcomponents of part 1.

From the example above, for part d, the parent components are part A and part 1 and hence the cross reference table will consist of two records with part d as Part Composition as seen in Table 3 and Table 4. The increase in records is illustrated in an example with only three levels and six elements. Consider the scenario when hundreds of levels and several thousand parts are considered as in the case of a traditional assembly of a complete product, the size of the cross reference table will increase exponentially to
millions of records. Such large cross reference tables are tedious manage each time an entity is added, removed or update.

<table>
<thead>
<tr>
<th>Part No</th>
<th>Part Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Product 1</td>
</tr>
<tr>
<td>A</td>
<td>Sub-assembly A</td>
</tr>
<tr>
<td>B</td>
<td>Sub-assembly B</td>
</tr>
<tr>
<td>d</td>
<td>Component d</td>
</tr>
<tr>
<td>e</td>
<td>Component e</td>
</tr>
</tbody>
</table>

Table 4: BOM Cross Reference Table

<table>
<thead>
<tr>
<th>Part No</th>
<th>Part Composition</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>c</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>d</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>e</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>c</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>e</td>
<td>3</td>
</tr>
</tbody>
</table>

Though RDBMSs can ultimately be used to build a database with recursive relationships, the challenge thereafter is recursive querying, the actual process of accessing and querying data through these recursive relationships [16]. This process of querying through recursive data structures is not supported by SQL.

2.3 Recursive Querying

The support for recursive queries in current query languages is limited and lacks theoretical foundations. Recursive queries are required for many tasks of database
applications and there is a constant need to use ad hoc queries or programs to address such structures. Even so, recursion is not supported in the SQL standards (SQL-89 and SQL-92) [17]. Since the standards do not support recursive queries, each relational DBMS and in particular Oracle and DB2 each implement them differently.

In some cases recursion can be substituted with *iteration* but this implies much lower programming level and less elegant problem specification. There is also the possibility that iteration could be a costly solution requiring complex code and program maintenance.

There are three established approaches to recursive querying [18];

- Extending SQL through the use of the *Transitive Closure Operator*
- *Least Fixed Point* equation systems
- *Recursive procedures* and views

The transitive closure operator is considered for the access control application since it, like many other recursive problems, can be reduced to a transitive closure problem [19]. For example, algorithms operating on genealogical trees, processing BOM data structures and operating on various kinds of networks. However, in relational databases implementing the transitive closure operator meets some non-trivial problems;

  - The transitive operator cannot be expressed in relational algebra and hence extensions have been proposed [20]
  - The computational power of the transitive closure operator may be insufficient [19]
  - Calculation of the transitive closure leads to performance problems [21]
The second approach is utilized in recursive tasks that cannot be expressed using the transitive closure operation, but can be expressed using a fixed point equation system [20]. In order to solve a recursive task, the least fixed point of an equations system must be found. This fixed point equation is used to perform the same computations as a transitional closure [18]. This approach’s biggest challenge is the lack of an efficient methodology supporting the developers of applications in the transition from business conceptual models to fixed point equations.

Since transitive closure operator and fixes point equations have their shortcomings, the alternate recursive procedures and functions approach was explored.

2.3.1 Recursive Procedures and Functions

The use of recursive procedures and functions is probably the most common approach to solving recursive query problems. A recursive procedure contains direct or indirect calls to itself. Recursive functions are often effective, efficient and elegant solutions and are commonly used in most programming languages. This is done by expressing a problem on a bigger scale as a multiple smaller scale solvable problems [18].

However, actual implementation of a recursive structures into a programming language function would require some kind of function output such as linked lists, since it is the easiest collection to implement. However database programming languages such as PL/SQL or Transact SQL are severely limited in recursive data processing and often these processes are outsourced to an external database management system utilizing programming languages such as Java or C++. There are several drawbacks that are
encountered when relational databases are utilized with modern programming languages. These are:

- The database is not seamlessly integrated with the programming language and they have to use database access APIs such as ODBC or JDBC.
- Databases and programming language type systems are usually different resulting in impedance mismatch, i.e. the need to convert data types during the processing.
- Output of programming languages cannot be bulk data in contrast to the requirements of queries.
- The actual processing of data takes place outside the database and hence there are issues that endanger the stability and integrity of the database.

The drawbacks and other factors considered, the best solution seems to be completely removing the relational database part and replacing it with high level programming language such as C++ or Java with the additional ability to permanently store and organize data values and facilitate the process of defining, constructing, querying and manipulating them. Such a solution is possible through the use of object oriented database management systems.

2.4 Object Oriented Databases – An Overview

Object oriented databases store their data in the form of objects which are instances of a class defined in a programming language. An object is a uniquely identifiable entity that contains attributes that describe the characteristics and state of the entity, and methods that describe the behaviors associated with their objects. When each
object is created, it is assigned a unique, independent and system generated Object Identifier (OID) which ensures that the object is uniquely identifiable [11].

Objects that have the same attributes and the same methods are grouped into classes. The attributes and methods are defined for each class and are not repeated for each object.

The concept of objects is derived from the object oriented programming languages where objects are considered to be transient and exist only during the program execution and are destroyed thereafter. The principle behind object oriented databases is to extend the life of an object and make them persist. Such objects are called persistent objects. Persistent objects are stored permanently and indefinitely beyond the termination of the program and can be retrieved at a later time and shared by peer programs [19].

Object oriented database management systems (OODBMS) boast the ability to overcome several of the RDBMSs’ shortfalls by empowering the system designer to specify both the data structure and the desired operations. As object oriented programming languages are used more widely in designing and implementing applications, OODBMS are preferred to enable seamless integration of the database management system and other information system applications.

Relationships, which are semantic constructs in relational databases, are not as well supported in OODBMS as they are in relational databases. However this challenge can be overcome since it is possible to program relationships using existing object-oriented constructs such as pointers [22].

A consortium of object oriented DBMS vendors and users, ODMG, proposed a standard model for OODBMS. Initially the ODMG 93 was introduced and then it was
revised to ODMG 2.0. The ODMG standard is made up of several parts; the object model, the object definition language (ODL), the object query language (OQL) and the bindings. It normally takes several years for a formal procedure for standards to be approved and the ODMG standard is still in the development and revising stages.

2.5 An Object Oriented Database Solution to Recursive Querying

The challenge in the centralized access control database system is the problem of efficiently and effectively querying recursive data structures to implement zone management. The use of object oriented database systems allows the seamless integration of powerful object oriented programming languages and therefore allows object oriented database systems to borrow concepts from object oriented programming languages. Concepts such as graph theory and link lists can be directly implemented into the database system [23].

A graph is a collection of nodes and edges. By considering a recursive data structure as a hierarchical tree, graph theory can be used to represent the structure of a recursive relationship. The data structure can be formally defined as a pair \( \{u,a\} \) where \( u \) denotes the objects of the zone class and a \( \{(mu,mv):mu,mv \in u\} \) represents the relationships between the objects. Using such a graph implementation, a link-list based data structure can be generated to model the entire database as shown in Figure 2.3. Through the use of directed and undirected graphs, unary relationships can be accurately modeled.
Recursive querying is achieved through *tree traversal procedures* [15] that are recursive in nature. The traversal of a tree is a process that enumerates or visits each node. Further, tree traversal concepts such as *tree pruning, path enumeration* and *path aggregation* may be used.

The objects of the database system are structured in the form of a tree. During the process of tree traversal, if the procedure returns to already traversed node, the tree is considered to have cycles and such a graph is considered a re-convergent graph. Since the area access control database system described in this thesis is not a re-convergent graph structure, the *path enumeration* method of recursive querying is employed versus the *path aggregation* method.

Path enumeration is a form of *traversal recursion*, which is a technique for traversing a graph starting from a node or set of nodes using the process of recursion. Path enumeration involves finding all nodes that may be reached from one or more starting nodes. This thesis employs a preorder path enumeration process [15]. In addition
to enumerating the traversed nodes of the database structure, the path enumeration process facilitates the ability to perform operations on each node traversed and derive information from the sub-trees rooted at each node during the traversal. The calculations of the privilege factor ($z_{cost}$), which will be discussed in Section 3.4.1, require this feature.

**2.6 An Object Oriented Implementation**

The object oriented database implementation starts with a data model. Using the ER diagram in Figure 2.1, a new data model can be designed for an equivalent object-oriented database. Unified Modeling Language (UML) is widely recognized as a general purpose visual modeling language used to represent object oriented databases [16,14]. It consists of diagramming tools which allows UML to capture information about the static structure and dynamic behavior of a system and facilitates the development and implementation of large scale software. A UML representation of the access control database system is shown in Figure 2.4.
The UML model shown in Figure 2.4 consists of *zone*, *personnel* and *transaction* classes. The objects of the *zone* class each represent a physical zone in the installation. These objects each contain data values that correspond to the attributes of each zone such as the zone id, zone name and the corresponding privilege cost. A privilege cost is assigned to each zone during the database design stage and they are used to select a path when multiple paths exist to reach a target zone by comparing their values. The use of the privilege cost is explained in detail in Section 3.4.1. Likewise, the objects of the
personnel class correspond to the each person enrolled in the area access control system and attributes pertaining to them.

Once a data model is developed, the object oriented database is defined using object definition language (ODL). ODL is an equivalent language to SQL except that it is specifically designed only to define the schema, which is the structure of the database and its constituents [11]. An empty database shell is generated by compiling the database schema file. The empty database is then ready for use by a compatible object oriented database management system. Queries are executed using the object query language (OQL) which is the equivalent of SQL in relational databases.

By using link-lists and collections or arrays of pointers, it is possible to implement all relationships including recursive relationships. This is done by specifying one or more pointers within the class definition (schema) that corresponding to each relationship that is required. When an object is related to another object, one object’s corresponding relationship pointer, points to the address of the object that it is related with. Through the use of multiple arrays, a one-to-many relationship may be built. In this method there is a transitive chain of relationships that is built and this chain can be traversed while querying in order to obtain the desired object. With the use of this type of implementation, it is immaterial if the objects being related are objects of the same class or not. Each object has a unique OID and they are considered individual and independent entities.
3. OBJECT ORIENTED AREA ACCESS CONTROL DATABASE SYSTEM

IMPLEMENTATION

This chapter describes the design and implementation of the centralized object
oriented area access control database system. The proposed approach supports recursive
data structures, unary relationships and recursive querying. FastObjects is chosen for the
access control database application based on the characteristics and features that the
product offers including a graphic user interface (GUI) [24].

3.1 Data model and Schema Generation

The preliminary step to obtain an object oriented database is a schema. A schema
is generated from a data model. The data model illustrated using UML in Figure 2.4 is
implemented for the area access control database system. In order to generate a schema,
the data model must be represented in an object definition file which is coded in an object
definition language. The object definition file which represents the schema is then
compiled using a schema compiler to obtain the empty object oriented database. The tool
used to define the schema is a text editor such as Windows Notepad.

3.2 Zone, Person and Transaction Class Definitions

To obtain the object definition file used to generate the schema and thereafter
compiled into the database, all the classes and their corresponding entities must be
finalized and defined. The three classes that need to be defined as per the data model are zone, person and transaction.

The zone class is the class that contains unary relationships and requires recursive query functionality, and is defined first. All other classes and relationships of the database are supplemented thereafter.

Figure 3.1 shows an ODL of the schema of the three classes corresponding to the area access control database system.

```
#include <ptstring.hxx>

persistent class zone
{
public:
    int z_id;
    PtString z_name;
    int z_cost;
    lset<zone*> shares_access_point;
};

persistent class person
{
public:
    int id;
    PtString firstname;
    PtString lastname;
    PtString title;
    lset<zone*> targetzone;
};

persistent class transaction
{
public:
    int t_id;
    PtString t_time;
    PtString t_date;
    zone * fromzone;
    zone * tozone;
    person * t_person;
};
```

Figure 3.1: ODL file for Area Access Control System’s Object Oriented Database
3.2.1 Zone Class

The definition of the zone class consists of two attributes, z_id and z_name which correspond to the zone’s identification number and zone name respectively. A privilege-cost factor, z_cost is introduced to allow the selection of appropriate path when more then one path is possible to the same target zone. A privilege-cost is assigned to each zone based on the level of security that is required for the zone and the privilege level that personnel requiring access to the zone are expected to have. By choosing a path with minimum costs, the least privilege path is obtained. A set of pointers lset<zone*>shares_access_with, corresponds to a set of recursive relationships that each object of the zone class may possess. The use of a set of pointers allows the number of relationships for each object to be unique and dynamic rather then specifying a single common number of pointers for all objects.

3.2.2 Person Class

The person class consists of a person identification number attribute, id, name attributes, firstname and lastname, a job title attribute, title and a set of pointers, lset<zone*>target_zone. The target-zone pointer set is used to assign one or several target zones to each person. Since the number of target zones varies with each person, a set of pointers is used and hence a maximum number specification is not required.

3.2.3 Transaction Class

The transaction class consists of three data attributes and three pointers attributes. The data attributes consist of the unique transaction identification number, a timestamp
attribute and a date-stamp attribute. The pointer attributes consist of pointers pointing to
the objects of the zone class that the person exited from and entered into. These two
pointers are the fromzone and tozone attributes. The third pointer is the person pointer
which points to the corresponding object of the person class referring to the person whose
movement is being recorded.

3.3 Database Setup and Population

The schema in Figure 3.1 is compiled using FastObjects compiler. An empty
database and header files corresponding to the database are generated by the OODBMS.
By including these header files into a C++ program, all database operations and
functionality is achieved through the use of standard internal methods specified by the
OODBMS that can be called from the C++ program. The empty object oriented database
is populated by either using C++ (using standard functions) or through the graphical user
interface (GUI) of the OODBMS.

Once the database is populated, C++ code is written to implement the zone
management and personnel tracking functionality. The C++ code used to traverse through
the recursive relationships of the objects of the zone class is shown in Appendix 1.

3.4 C++ Implementation

The procedures described in the preceding sections require implementation in an
object oriented programming language. FastObjects exclusively supports C++ and Java
and C++ was selected as the language of implementation. Once the procedures are
accurately implemented, the program is executed and the various functionalities are
tested and results are verified. The C++ procedures that are necessary to perform the zone management and personnel tracking is described in the following sections.

### 3.4.1 Zone Management

The implementation of the zone management feature consists of three procedures, *locateperson*, *direct_zones* and *zone_mgmt*. Once FastObjects services and internal functions are initialized, the *locateperson* function locates the person object in the person class extent. An *extent* of a class is defined as an object set that contains all the persistent objects of a class. Once the desired person object is found, its object id (OID) is used to call the *direct_zones* function.

The *direct_zones* function is used to traverse through the target zones assigned to each person object. For each of these target zones, the preliminary zones are required. In order to obtain the zones requiring preliminary access, a recursive function, *zone_mgmt* is called by passing the target zone’s OID.

In the *zone_mgmt* function, each target zone object’s recursive relationship set *shares_access_point* is traversed and all zones that participate in the recursive relationship are identified. The zone objects participating in the relationship are the zones that require preliminary access. There are situations where more than a single path is possible to approach the same target zone which means there are two paths to traverse from that node in the tree. In such a scenario, the privilege cost factor (*z_cost*) of the next zone in each path is compared. The function is programmed to select the path through a zone with a lower privilege cost.
To illustrate the least cost privilege concept, consider the case shown in Figure 1.1. To access Zone 3, it is sufficient to access Zone 1 and Zone 2. However, during the traversal process the program will encounter the existence of an alternate path through Zone 5 due to the presence of an entry (F). In order to ensure the program does not select the alternate path, the \( z_{\text{cost}} \) values of Zone 2 and Zone 5 are compared during the traversal process. If Zone 5 is assigned a higher privilege cost than Zone 2 during the database design, the path through Zone 5 is disqualified from consideration and the path with the lower privilege, Zone 2, is selected.

3.4.2 Personnel Tracking

The personnel tracking functionality implemented in the thesis application consists of three features:

- The ability to determine the location of a person at a particular time.
- The ability to determine all the zones through which a person moved during a time period.
- The ability to determine all persons with who a specific person interacted during a time period and the zones in which each interaction took place.

The three tracking features listed above are selected to cover the basic tracking functions that may be extended for other tracking features. For example, the ability to determine the location of a person during a range of time can be modified to determine all persons’ movement through a specific zone during a time period by using the zone object rather than the person object as the object of interest. Other tracking features can also be
incorporated based on the same concepts of the features listed above by changing the object of interest and the tracking factors.

Four procedures are used to implement the required personnel tracking features. The procedures are, person\_at\_specific\_time, person\_between\_times, interaction\_between\_times and check interaction.

The person\_at\_specific\_time function takes a person identification number (person\_id) and the time (t) as inputs. The procedure iterates through the objects of the transaction class analyzing the transaction objects’ t\_person pointer. The identification number (id) of the person object related to the transaction object through the t\_person pointer is compared with the input (person\_id). For each object matching the person\_id, the transaction times are checked. The first transaction involving the person after the input time (t) is analyzed. The fromzone attribute of this transaction object provides the location of the person at the input time.

The person\_between\_times function takes the person\_id and the two time limits (t1 and t2) as inputs. This function operates just as the person\_at\_specific\_time. For each object whose corresponding person object matches the person\_id, the transaction times are compared to obtain all the transactions within the time limits. By using the fromzone attribute from the first transaction within the range and the tozone attributes of the rest of the transactions within the time range, the movement of the person through the various zones is tracked and all the zones are obtained.

The interaction\_between\_times procedure is used to obtain all interactions between a specific person and all other persons during their movement through the various zones between two time limits. The inputs for this procedure are the person\_id,
and the two time limits (t1 and t2). This procedure identifies the time intervals spent by the person within each zone during the time limits. This is done by iterating through the transaction table and comparing the person_id, and using the fromzone and tozone attributes as in the person_between_times function. Once the time limits spent within each individual zone during the desired time range are obtained, the zone_id, and new time limits (tnew1 and tnew2) for each interval of time spent in a zone are sent as inputs to another procedure called check_interaction.

The check_interaction procedure iterates through the objects of the transaction table and checks the fromzone and tozone attributes for all transaction objects lying within the time limits sent from the interaction_between_times procedure. When there match in the zones and times, there is a possibility that some interaction may have taken place. Using the id of the person corresponding to the transaction objects that match, each person who may have interacted with the individual in question is listed.
4. RESULTS

This chapter describes the results of the area access control program’s implementation. The four functionalities that are implemented in the area access control system are;

- Zone Management: Determine all direct target zones assigned to a person and then the zones that preliminary require access to access the target zone.
- Tracking: Determine a person’s location at specific time.
- Tracking: Determine a person’s movement during a specific time period.
- Tracking: Determine a person’s interaction with other persons during a specific time period.

To illustrate the implementation and compare results, the hypothetical case shown in Figure 1.1 is shown below as Figure 4.1 and its topography is used.
4.1.1 Zone Management

The input for the zone management feature is a person’s identification and the outputs are the direct target zones assigned and the preliminary access zones. The program output below shows the result of the zone management feature for a person (123-46-789) who requires access to target zones 4 and 5. From Figure 4.1, the preliminary zones that the person needs access to can be determined to be zones 3, 2 and
1 for zone 4, and zones 2 and 1 for zone 5. The program output shown in Figure 4.2 is checked with the case shown in Figure 4.1. An inspection of the case indicates that the result of the zone management feature for target zones 5 and 4 is the list of zones shown in Figure 4.2, hence the implementation of zone management.

![Figure 4.2: Program Output for Zone Management](image)

4.1.2 Tracking

The tracking features are performed by analyzing objects of the transactions class. Figure 4.3 shows the program output as the location of a person (987-65-4321) at time 10:00:00 in the area access control system. The result may be compared with the corresponding relational transaction table. As seen from the highlighted record in the
Transaction Table shown in Table 5, the location of person 987-65-4321 at 10:00:00 is Zone 3.

![Program Output for Tracking: Location of Person at Specific Time](image)

Figure 4.3: Program Output for Tracking: Location of Person at Specific Time
Table 5: Records Corresponding to the Location of Person (987-65-4321) at 10:00:00

<table>
<thead>
<tr>
<th>tid</th>
<th>tdate</th>
<th>ttime</th>
<th>zfrom</th>
<th>zto</th>
<th>pid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6/27/2005</td>
<td>6:54:00 AM</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6/27/2005</td>
<td>6:58:00 AM</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>6/27/2005</td>
<td>8:22:00 AM</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>6/27/2005</td>
<td>8:24:00 AM</td>
<td>1</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6/27/2005</td>
<td>8:29:00 AM</td>
<td>2</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6/27/2005</td>
<td>8:59:00 AM</td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>6/27/2005</td>
<td>9:02:00 AM</td>
<td>1</td>
<td></td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>6/27/2005</td>
<td>9:03:00 AM</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>6/27/2005</td>
<td>9:04:00 AM</td>
<td>1</td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>6/27/2005</td>
<td>9:05:00 AM</td>
<td>2</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>6/27/2005</td>
<td>9:07:00 AM</td>
<td>2</td>
<td></td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>6/27/2005</td>
<td>9:13:00 AM</td>
<td>3</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>6/27/2005</td>
<td>9:21:00 AM</td>
<td>3</td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>6/27/2005</td>
<td>9:26:00 AM</td>
<td>4</td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>6/27/2005</td>
<td>10:01:00 AM</td>
<td></td>
<td></td>
<td>15</td>
</tr>
<tr>
<td>16</td>
<td>6/27/2005</td>
<td>10:08:00 AM</td>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>17</td>
<td>6/27/2005</td>
<td>11:12:00 AM</td>
<td></td>
<td></td>
<td>17</td>
</tr>
<tr>
<td>18</td>
<td>6/27/2005</td>
<td>11:26:00 AM</td>
<td></td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>6/27/2005</td>
<td>11:34:00 AM</td>
<td></td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>20</td>
<td>6/27/2005</td>
<td>11:55:00 AM</td>
<td></td>
<td></td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>6/27/2005</td>
<td>12:22:00 PM</td>
<td></td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>6/27/2005</td>
<td>12:28:00 PM</td>
<td></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td>23</td>
<td>6/27/2005</td>
<td>12:31:00 PM</td>
<td></td>
<td></td>
<td>23</td>
</tr>
<tr>
<td>24</td>
<td>6/27/2005</td>
<td>12:41:00 PM</td>
<td></td>
<td></td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>6/27/2005</td>
<td>12:47:00 PM</td>
<td></td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>26</td>
<td>6/27/2005</td>
<td>12:48:00 PM</td>
<td></td>
<td></td>
<td>26</td>
</tr>
</tbody>
</table>

Figure 4.4 shows the program output as the movement of a person (123-45-6789) from time 09:00:00 to 11:00:00 in the area access control system. The results may be compared with the corresponding relational transaction table. As seen from the highlighted records in the Transaction Table shown in Table 6, the locations though which person 123-45-6789 moved between 09:00:00 and 11:00:00 are zone 1, zone 2, zone 3 and zone 5.
Figure 4.4: Program Output for Tracking: Location of Person between Two Time Limits
Table 6: Records Corresponding to the Location of Person (123456789) between 9:00:00 and 11:00:00

<table>
<thead>
<tr>
<th>tid</th>
<th>tdate</th>
<th>time</th>
<th>zfrom</th>
<th>zto</th>
<th>pid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6/27/2005</td>
<td>6:54:00 AM</td>
<td></td>
<td></td>
<td>1 678912346</td>
</tr>
<tr>
<td>2</td>
<td>6/27/2005</td>
<td>6:58:00 AM</td>
<td>1</td>
<td>2</td>
<td>678912346</td>
</tr>
<tr>
<td>3</td>
<td>6/27/2005</td>
<td>8:22:00 AM</td>
<td></td>
<td></td>
<td>1 543216789</td>
</tr>
<tr>
<td>4</td>
<td>6/27/2005</td>
<td>8:24:00 AM</td>
<td>1</td>
<td></td>
<td>2 543216789</td>
</tr>
<tr>
<td>5</td>
<td>6/27/2005</td>
<td>8:29:00 AM</td>
<td></td>
<td></td>
<td>3 543216789</td>
</tr>
<tr>
<td>6</td>
<td>6/27/2005</td>
<td>8:59:00 AM</td>
<td></td>
<td></td>
<td>1 23456789</td>
</tr>
<tr>
<td>7</td>
<td>6/27/2005</td>
<td>9:02:00 AM</td>
<td>1</td>
<td>2</td>
<td>1 23456789</td>
</tr>
<tr>
<td>8</td>
<td>6/27/2005</td>
<td>9:03:00 AM</td>
<td></td>
<td></td>
<td>1 987654321</td>
</tr>
<tr>
<td>9</td>
<td>6/27/2005</td>
<td>9:04:00 AM</td>
<td></td>
<td>1</td>
<td>2 987654321</td>
</tr>
<tr>
<td>10</td>
<td>6/27/2005</td>
<td>9:05:00 AM</td>
<td>1</td>
<td></td>
<td>2 3 123456789</td>
</tr>
<tr>
<td>11</td>
<td>6/27/2005</td>
<td>9:07:00 AM</td>
<td></td>
<td>2</td>
<td>3 987654321</td>
</tr>
<tr>
<td>12</td>
<td>6/27/2005</td>
<td>9:13:00 AM</td>
<td></td>
<td></td>
<td>2 123456789</td>
</tr>
<tr>
<td>13</td>
<td>6/27/2005</td>
<td>9:21:00 AM</td>
<td></td>
<td>3</td>
<td>4 987654321</td>
</tr>
<tr>
<td>14</td>
<td>6/27/2005</td>
<td>9:26:00 AM</td>
<td></td>
<td>4</td>
<td>3 987654321</td>
</tr>
<tr>
<td>15</td>
<td>6/27/2005</td>
<td>10:01:00 AM</td>
<td></td>
<td>3</td>
<td>4 987654321</td>
</tr>
<tr>
<td>16</td>
<td>6/27/2005</td>
<td>10:08:00 AM</td>
<td></td>
<td>4</td>
<td>3 987654321</td>
</tr>
<tr>
<td>17</td>
<td>6/27/2005</td>
<td>11:12:00 AM</td>
<td></td>
<td>5</td>
<td>6 12346789</td>
</tr>
<tr>
<td>18</td>
<td>6/27/2005</td>
<td>11:26:00 AM</td>
<td></td>
<td>3</td>
<td>4 987654321</td>
</tr>
<tr>
<td>19</td>
<td>6/27/2005</td>
<td>11:34:00 AM</td>
<td></td>
<td>3</td>
<td>5 123456789</td>
</tr>
<tr>
<td>20</td>
<td>6/27/2005</td>
<td>11:56:00 AM</td>
<td></td>
<td>4</td>
<td>3 987654321</td>
</tr>
<tr>
<td>21</td>
<td>6/27/2005</td>
<td>12:22:00 PM</td>
<td></td>
<td>3</td>
<td>2 987654321</td>
</tr>
<tr>
<td>22</td>
<td>6/27/2005</td>
<td>12:28:00 PM</td>
<td></td>
<td>2</td>
<td>1 987654321</td>
</tr>
<tr>
<td>23</td>
<td>6/27/2005</td>
<td>12:31:00 PM</td>
<td></td>
<td>1</td>
<td>987654321</td>
</tr>
<tr>
<td>24</td>
<td>6/27/2005</td>
<td>12:41:00 PM</td>
<td></td>
<td>3</td>
<td>2 123456789</td>
</tr>
<tr>
<td>25</td>
<td>6/27/2005</td>
<td>12:47:00 PM</td>
<td></td>
<td>2</td>
<td>1 123456789</td>
</tr>
<tr>
<td>26</td>
<td>6/27/2005</td>
<td>12:48:00 PM</td>
<td></td>
<td>1</td>
<td>123456789</td>
</tr>
</tbody>
</table>

The final personnel tracking feature requires the program to determine all the interactions between a specific person and other persons based on their movements. Figure 4.5 illustrates the program output of this feature. The inputs are a person’s identification (543-21-6789) and two time limits (08:30:00 and 10:00:00). The program generates all interactions that may have taken place with the individual during the time interval. The resulting output is compared with the corresponding relational transaction table. As seen from the highlighted records in the Transaction Table shown in Figure 4.8,
the person 123-45-6789 interacted with 987-65-4321 in zone 2 and thereafter multiple times in zone 3, between 09:00:00 and 11:00:00.

Figure 4.5: Program Output for Tracking: Personnel Interaction between Two Time Limits
Table 7: Records Corresponding to the Personnel Interaction between (123-45-6789) between 9:00:00 and 11:30:00

<table>
<thead>
<tr>
<th>tid</th>
<th>tdate</th>
<th>time</th>
<th>zfrom</th>
<th>zto</th>
<th>pid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6/27/2005</td>
<td>8:54:00 AM</td>
<td></td>
<td></td>
<td>1678912346</td>
</tr>
<tr>
<td>2</td>
<td>6/27/2005</td>
<td>8:58:00 AM</td>
<td>1</td>
<td></td>
<td>2678912346</td>
</tr>
<tr>
<td>3</td>
<td>6/27/2005</td>
<td>8:22:00 AM</td>
<td></td>
<td></td>
<td>1543216789</td>
</tr>
<tr>
<td>4</td>
<td>6/27/2005</td>
<td>8:24:00 AM</td>
<td>1</td>
<td></td>
<td>2543216789</td>
</tr>
<tr>
<td>5</td>
<td>6/27/2005</td>
<td>8:29:00 AM</td>
<td></td>
<td>2</td>
<td>3543216789</td>
</tr>
<tr>
<td>6</td>
<td>6/27/2005</td>
<td>8:59:00 AM</td>
<td></td>
<td>1</td>
<td>123456789</td>
</tr>
<tr>
<td>7</td>
<td>6/27/2005</td>
<td>9:02:00 AM</td>
<td></td>
<td>1</td>
<td>123456789</td>
</tr>
<tr>
<td>8</td>
<td>6/27/2005</td>
<td>9:03:00 AM</td>
<td></td>
<td></td>
<td>19987654321</td>
</tr>
<tr>
<td>9</td>
<td>6/27/2005</td>
<td>9:04:00 AM</td>
<td></td>
<td>1</td>
<td>2987654321</td>
</tr>
<tr>
<td>10</td>
<td>6/27/2005</td>
<td>9:05:00 AM</td>
<td></td>
<td>2</td>
<td>3123456789</td>
</tr>
<tr>
<td>11</td>
<td>6/27/2005</td>
<td>9:07:00 AM</td>
<td></td>
<td>2</td>
<td>3987654321</td>
</tr>
<tr>
<td>12</td>
<td>6/27/2005</td>
<td>9:13:00 AM</td>
<td></td>
<td>3</td>
<td>6123456789</td>
</tr>
<tr>
<td>13</td>
<td>6/27/2005</td>
<td>9:21:00 AM</td>
<td></td>
<td>3</td>
<td>4987654321</td>
</tr>
<tr>
<td>14</td>
<td>6/27/2005</td>
<td>9:26:00 AM</td>
<td></td>
<td>4</td>
<td>3987654321</td>
</tr>
<tr>
<td>15</td>
<td>6/27/2005</td>
<td>10:01:00 AM</td>
<td></td>
<td>3</td>
<td>4987654321</td>
</tr>
<tr>
<td>16</td>
<td>6/27/2005</td>
<td>10:38:00 AM</td>
<td></td>
<td>4</td>
<td>3987654321</td>
</tr>
<tr>
<td>17</td>
<td>6/27/2005</td>
<td>11:12:00 AM</td>
<td></td>
<td>5</td>
<td>3123456789</td>
</tr>
<tr>
<td>18</td>
<td>6/27/2005</td>
<td>11:26:00 AM</td>
<td></td>
<td>3</td>
<td>4987654321</td>
</tr>
<tr>
<td>19</td>
<td>6/27/2005</td>
<td>11:34:00 AM</td>
<td></td>
<td>3</td>
<td>5123456789</td>
</tr>
<tr>
<td>20</td>
<td>6/27/2005</td>
<td>11:56:00 AM</td>
<td></td>
<td>4</td>
<td>3987654321</td>
</tr>
<tr>
<td>21</td>
<td>6/27/2005</td>
<td>12:22:00 PM</td>
<td></td>
<td>3</td>
<td>2987654321</td>
</tr>
<tr>
<td>22</td>
<td>6/27/2005</td>
<td>12:26:00 PM</td>
<td></td>
<td>2</td>
<td>1987654321</td>
</tr>
<tr>
<td>23</td>
<td>6/27/2005</td>
<td>12:31:00 PM</td>
<td></td>
<td>1</td>
<td>987654321</td>
</tr>
<tr>
<td>24</td>
<td>6/27/2005</td>
<td>12:41:00 PM</td>
<td></td>
<td>3</td>
<td>2123456789</td>
</tr>
<tr>
<td>25</td>
<td>6/27/2005</td>
<td>12:47:00 PM</td>
<td></td>
<td>2</td>
<td>1123456789</td>
</tr>
<tr>
<td>26</td>
<td>6/27/2005</td>
<td>12:48:00 PM</td>
<td></td>
<td>1</td>
<td>123456789</td>
</tr>
</tbody>
</table>
5. CONCLUSIONS AND FUTURE WORK

5.1 Conclusion

This thesis explores the use of the object oriented database management systems (OODBMS) for the area access control database system. An OODBMS adequately addresses both the need for supporting unconventional data types and the ability to support recursive data, establish unary relationships and perform recursive queries. Chapter 3 and 4 illustrate the implementation and results of the object oriented area access control database system. A Relational DBMS approach, their inadequacies and limitations are described in Chapter 2.

The object oriented area access control system allows the implementation of zone management. Zone management is a feature by which the database structure alone may be utilized to determine the path from outside the installation to a target zone within the installation and all the zones in between these two points that require prior access by a user. By using path enumeration, the area access control program is also able to make a choice between multiple paths based on predetermined policies such as the least privilege cost (z_cost) policy as implemented in this thesis.

Zone management can only be implemented if the database supports unary relationships and recursive querying. This limitation arises because zone entities in the database need to be related to other zone entities in order to implement zone management. Since records of the same table cannot be related to each other, relational
databases require the use of cross-reference tables. The size of a cross-reference table increases exponentially as the number of entities participating in the recursive relationship increases. Cross-reference tables are not necessary when using object oriented database management systems since objects of the same class may be related to each other through the use of unary relationships. Establishing such recursive structures through unary relationships requires recursive procedures to query, and enumerate through all the entities. Such recursive procedures may be seamlessly integrated with object oriented databases using object oriented programming languages.

Once object oriented database management systems are selected, object oriented programming languages may be integrated with the database to implement tracking procedures. If the database is a relational database, the tracking procedures would be separate programs which would have to be integrated into the database through access APIs such as ODBC or JDBC. Integration between relational databases and object oriented programming languages may cause type mismatches or endanger the database since the actual processing of data takes place outside the database. This thesis uses object oriented databases to overcome these difficulties.

The tracking features implemented in this thesis allows the administrator of the object oriented database management system, to check the movement of persons within the installation at any given time and all the interactions that the person has had with other persons within the installation. The results of the tracking features are shown in Chapter 4.

The use of objects and the ability to define the data types within the definition of a class allows the use of unconventional data types. A user may specify unique user
defined data types within the class definition and thereafter new objects which are instances of the class are able to ‘persist’ these data types. User defined data types such as image and graphic data, sound data or other data types would otherwise not be persistent and cannot easily be stored with relational databases. The use of biometric systems in area access control systems which utilize unique data types brings about the need for databases that support unconventional data types.

These contributions conclude that the use of an object oriented database management system allows the implementation of an area access control database system with zone management and personnel tracking features.

5.2 Working Scheme of the Object Oriented Area Access Database System

A working scheme of the object oriented area access control system is illustrated in Figure 5.1 and described in this section.
Figure 5.1: Working Scheme of the Centralized Object Oriented Area Access Control Database System

The object oriented area access control database system is located in a central location either inside the installation or elsewhere. The database system is connected to several access control devices, each providing access to different zones within the installation, through a Local Area Network (LAN) or Wide Area Network (WAN). Each access control device such as a swipe card reader or scanner references a table that contains records of all personnel authorized to enter through that access control device.

The enrollment process is done in the central database. When a person is enrolled into the installation’s database, a new person object is created. The person’s object
identification number and the target zones are specified into the enrollment program. This program uses the zone management feature to automatically determine a list of all zones that the person requires prior access to. At pre-determined time intervals, all new enrollments or updates to persons’ information is transmitted through the LAN/WAN network to each access control device that provides access to each zone on the persons’ zone management list. A new record for the person is created in each table corresponding to the access control devices. In this way the access control system ensures that all prior access is assigned. The use of individual tables at each access control device is preferred instead of live transmission through the LAN/WAN for each transaction since minimal communication ensures greater security and integrity of the central database. The use of local tables also ensures that the checking process is less dependent on network speeds and location of the central database.

Transactions may be recorded at a central location via transmissions through the LAN/WAN or a transaction table may be located at each access control device. If local transaction tables are used, at specific intervals, the transaction information is transmitted to the central database. It is advisable to transmit live transaction information via the LAN/WAN since the tracking feature requires up-to-date information on persons’ transactions. There is no significant time loss due to network load or database location since the transaction recording process does not affect the response of the access control device. By using the tracking procedures available with area access control system on the central transaction database, the required information may be obtained.
5.3 Future Work

Tasks that may be carried out to extend this thesis in the future include:

- Extend the features of personnel tracking to tracking vehicles, objects and animals within installations.

- Other database types may be explored to check if the Zone Management and Personnel Tracking features are possible.

- The use of central, distributed or hybrid databases may be explored and their effects on the features of the access control database system may be studied.
REFERENCES


Appendix A: C++ Program for Traversing Recursive Relations

```cpp
#include <poet.hxx>
#include "c:\aacs\aacs.hxx"
#include "c:\aacs\aacs.cxx"
#include <iostream.h>
#include <ptstring.hxx>
#include <stdio.h>
#include <string.h>
#include "stdafx.h"
#include <math.h>

//forward declarations
int locateperson(PtBase* pBase, person* per);
int direct_zones(PtBase* pBase, person* per2);
int zone_mgmt(PtBase* pBase, zone* pzone);
int trans_display(PtBase* pBase);
void personatspecifictime(PtBase* pBase);
void person_between_times(PtBase* pBase);
void interactions_between_times(PtBase* pBase);
void check_interaction(PtBase* pBase,double testtime,double double_value_of_time,int zone_id,int person_id);

// Main function
int main(int argc, char** argv)
{
    int err = 0;
    PtBase* pBase = (PtBase*) 0;
    // Initializes FastObjects services
    InitPOET(PtTransactionByThread, "BOM");
    #ifdef _WIN32
    PtString::SetDefaultCodeSet( PtCODESET_OS2 );
    #else
    PtString::SetDefaultCodeSet( PtCODESET_ANSI );
    #endif
    // Path Information to Locate Database
    err = PtBase::POET()->GetBase("LOCAL", "c:\aacs\base", pBase);
    if (err < 0)
    {
        cerr << "Could not open database. Error: " << err << "." << endl;
    }
    else
    {
        // Declare a transaction
        PtTransaction transaction;
        PtBase::POET()->SetCurrentTransaction( &transaction );
        err = transaction.RegisterResource( pBase );
        if (err < 0)
        {
            cerr << "Could not register transaction. Error: " 
            << err << "." << endl;
        }
        else
```

57
Appendix A (Continued)

```c
{ 
    zone* zon1 = (zone*) 0;
    person* person1 = (person*) 0; // defining a null pointer of type

    transaction.Begin();

    int optionvar;
    cout<<"\nPlease select option 1/2 below \n    1) Display Zone Management Feature\n    2) Display Personnel Tracking Feature\n Option Number ? ::\n";
    cin>>optionvar;

    if (optionvar==1)
    {
        // calling the function to locate the user specified person
        err = locateperson(pBase, person1);
    }
    else if (optionvar==2)
    {
        err = trans_display(pBase); // opens menu for personnel tracking
    }
    cout<<"\n\nEND OF PROGRAM\n\n";

    transaction.Commit();
    transaction.DeregisterResource( pBase );
}

PtBase::POET()->SetCurrentTransaction( 0 );
PtBase::POET()->UngetBase(pBase);
}

DeinitPOET();
return err;
}

// function to locate the user specified object by iterating through the person class AllSet
int locateperson(PtBase* pBase, person* per)
{
    int err = 0;
    zoneAllSet allzone(pBase);
    personAllSet allperson(pBase);
    int num=allperson.GetNum();
    int person_id;
    zone* zon2 = (zone*) 0;
    person* per1 = (person*) 0;

    cout<<"\nPlease give the Person ID for the component to find Permitted Zones: ";
    cin>>person_id;

    // finding the corresponding object
    for ( err = allperson.Get(per, 0, PtSTART);
            err >= 0;
            err = allperson.Get(per, 1, PtCURRENT) )
```

58
Appendix A (Continued)

```c
{    if (person_id==int(per->id))
    {        per1=per; //assigning the object found to the object sent to other functions
            break;
    }
    allperson.Unget(per);
}

    cout<>"\nPerson ID : " <<per1->id;
    err = direct_zones(pBase, per1); // calling the directzones function
    return err;
}

// This Function determines the target zones assigned to the person
int direct_zones(PtBase* pBase, person* per)
{
    int err=0;
    int test_no;
    zone* pzone = (zone*)0;
    for (int i = 0; ! (err=per->targetzone.Get(pzone, i, PtSTART)); ++i)
    {
        test_no=(int)pzone->z_id;
        cout<>"\n Target Zone :"<<test_no;
        cout<>"\n Zone Management :":;
        zone_mgmt(pBase,pzone);
    }
    return err;
}

//function to iterate through sub-assemblies and components to obtain their counts
int zone_mgmt(PtBase* pBase, zone* pzone)
{
    int err=0;
    int test_cost=100;
    int test_zoneid=0;
    int test_no;
    zone* nextzone = (zone*)0;

    //checking costs of each preliminary zone
    for (int j = 0; ! (err=pzone -> shares_access_point.Get(nextzone, j, PtSTART)); ++j)
    {
        if ((int)nextzone->z_cost<test_cost)
        {
            test_cost = (int)nextzone->z_cost;
            test_zoneid = (int)nextzone->z_id;
        }
    }

    //iterating through current element's constituents
    for (int i = 0; ! (err=pzone -> shares_access_point.Get(nextzone, i, PtSTART)); ++i)
    {
    }
```

59
Appendix A (Continued)

```c
    test_no=(int)nextzone->z_id;
//recursively calling the function in itself for each element till the basic components
    if (test_no == test_zoneid)
    {
        cout<"n"<test_no;
        zone_mgmt(pBase,nextzone);
    }
    return err;
}
//this function displays the options for tracking feature
int trans_display(PtBase* pBase)
{
    int transoptionvar;
    cout<"nPlease select from options below";
    cout<"n 1) Display location of Person at a specific time";
    cout<"n 2) Track movement of a person between two times";
    cout<"n 3) Determine all interactions with a Person between times";
    cout<"n 4) Exit";
    cout<"n Option Number? ::";
    cin>> transoptionvar;

    switch(transoptionvar)
    {
    case 1:
    {
        personatspecifictime(pBase);
        break;
    }
    case 2:
    {
        person_between_times(pBase);
        break;
    }
    case 3:
    {
        interactions_between_times(pBase);
        break;
    }
    default:
    {
        break;
    }
    }
    return 0;
}
// this function determines the location of a person at a specific time
void personatspecificertime(PtBase* pBase)
{
    transactionAllSet alltrans(pBase);
```
Appendix A (Continued)

```cpp
int num = alltrans.GetNum();
int err, hh, mm, ss;
int person_id;
//zone* zon2 = (zone*) 0;
transaction* tran = (transaction*) 0;
char transtimetest[8];
cout << "\nPlease give the Person ID:";
cin >> person_id;
cout << "\nPlease give the Time:";
cin >> transtimetest;
CString transtime = transtimetest;
double doubletime;
hh = atoi (transtime.Left(2));
mm = atoi (transtime.Mid(3, 2));
ss = atoi (transtime.Mid(6, 2));
doubletime = ((double)hh / 24) + ((double)mm / 1440) + ((double)ss / 86400);
t_counter = 0;
int zone_number = 0;

// finding the corresponding object
for ( err = alltrans.Get(tran, 0, PtSTART);
    err >= 0;
    err = alltrans.Get(tran, 1, PtCURRENT) )
{
    t_counter = t_counter + 1;
    CString transtimecheck = tran->t_time.StrGet();
double double_value_of_time = strtod(transtimecheck, NULL);
double value_of_time = double_value_of_time - 38530;
if (tran->t_person->id == person_id)
{
    if (double_value_of_time > doubletime)
    {
        zone_number = tran->fromzone->z_id;
        break;
    }
    else if (double_value_of_time == doubletime)
    {
        zone_number = tran->tozone->z_id;
        break;
    }
}
}

cout << "\nThe Person was in Zone :" << zone_number;
}

// this function determines the movement of a person between specific times
void person_between_times(PtBase* pBase)
{
    transactionAllSet alltrans(pBase);
    int num = alltrans.GetNum();
    int err, hh1, mm1, ss1, hh2, mm2, ss2;
```
int person_id;
transaction* tran = (transaction*) 0;
char transtimetest1[8];
char transtimetest2[8];
cout<<"\nPlease enter the Person ID: ";
cin>>person_id;
cout<<"\nPlease enter the Tracking Start Time: ";
cin>>transtimetest1;
CString transtime1 = transtimetest1;
cout<<"\nPlease enter the Tracking End Time: ";
cin>>transtimetest2;
CString transtime2 = transtimetest2;
double doubletime1;
double doubletime2;
int zone_number[10];
int t_counter=0;

// finding the corresponding object
for ( err = alltrans.Get(tran, 0, PtSTART);
    err >= 0;
    err = alltrans.Get(tran, 1, PtCURRENT) )
{
    CString transtimecheck = tran->t_time.StrGet();
double double_value_of_time = strtod(transtimecheck,NULL);
double_value_of_time = double_value_of_time - 38530;
if (tran->t_person->id == person_id)
{
    if ((t_counter==0)&&(double_value_of_time>doubletime1))
    {
        zone_number[t_counter]=tran->fromzone->z_id;
        t_counter++;
        zone_number[t_counter]=tran->tozone->z_id;
        t_counter++;
    }
    else if ((t_counter>0)&&(double_value_of_time<doubletime2))
    {
        zone_number[t_counter]=tran->tozone->z_id;
        t_counter++;
    }
}
Appendix A (Continued)

```cpp
int stopper = t_counter++;
cout<<"nThe person has moved through the following zones :";
for (int i=0;i<stopper;i++)
{
    cout<<"n"<<zone_number[i];
}
}

// this function determines the interactions between a person and other between specific times
void interactions_between_times(PtBase* pBase)
{
    transactionAllSet alltrans(pBase);
    int num=alltrans.GetNum();
    int err,hh1,mm1,ss1,hh2,mm2,ss2;
    int person_id;
    transaction* tran = (transaction*) 0;
    char transtimetest1[8];
    char transtimetest2[8];
    cout<<"n
Please enter the Person ID:";
    cin>>person_id;
    cout<<"n
Please enter the Tracking Start Time:";
    cin>>transtimetest1;
    CString transtime1 = transtimetest1;
    cout<<"n
Please enter the Tracking End Time:";
    cin>>transtimetest2;
    CString transtime2 = transtimetest2;
    double doubletime1;
    double doubletime2;
    hh1 = atoi (transtime1.Left(2));
    mm1 = atoi (transtime1.Mid(3,2));
    ss1 = atoi (transtime1.Mid(6,2));
    hh2 = atoi (transtime2.Left(2));
    mm2 = atoi (transtime2.Mid(3,2));
    ss2 = atoi (transtime2.Mid(6,2));
    doubletime1 = (((double)hh1/24)+((double)mm1/1440)+((double)ss1/86400));
    doubletime2 = (((double)hh2/24)+((double)mm2/1440)+((double)ss2/86400));

    int t_counter=0;
    double testtime = doubletime1;
    int zone_id;

    // finding the corresponding object
    for ( err = alltrans.Get(tran, 0, PtSTART);
        err >= 0;
        err = alltrans.Get(tran, 1, PtCURRENT) )
    {
        CString transtimecheck = tran->t_time.StrGet();
        double double_value_of_time = strtod(transtimecheck,NULL);
        double_value_of_time = double_value_of_time - 38530;
    }
```

if (tran->t_person->id == person_id)
{
    if ((t_counter==0)&&(double_value_of_time>doubletime1))
    {
        //zone_number[t_counter]=tran->fromzone->z_id;
        //zone_number[t_counter]=tran->tozone->z_id;
        zone_id=tran->fromzone->z_id;
        check_interaction(pBase,testtime,double_value_of_time,
        zone_id, person_id);
        testtime=double_value_of_time;
        t_counter++;
    }
    else if ((t_counter>0)&&(double_value_of_time<doubletime2))
    {
        //zone_number[t_counter]=tran->tozone->z_id;
        zone_id=tran->fromzone->z_id;
        check_interaction(pBase,testtime,double_value_of_time,
        zone_id, person_id);
        testtime=double_value_of_time;
        check_interaction(pBase,testtime,doubletime2, zone_id,
        person_id);
        t_counter++;
    }
}
}

// part of the interaction functionality
void check_interaction(PtBase* pBase,double testtime1,double testtime2,int zone_id,int person_id)
{
    transactionAllSet alltrans(pBase);
    transaction* tran = (transaction*) 0;
    int err;
    int t_counter=0;

    for ( err = alltrans.Get(tran, 0, PtSTART);
    err >= 0;
    err = alltrans.Get(tran, 1, PtCURRENT) )
    {
        CString transtimecheck = tran->t_time.StrGet();
        double double_value_of_time = strtod(transtimecheck,NULL);
        double_value_of_time = double_value_of_time - 38530;

        if ((int)tran->t_person->id != person_id)
        {
            if ((t_counter==0)&&(double_value_of_time>testtime1))
            {
                if (tran->fromzone->z_id==zone_id)
Appendix A (Continued)

```cpp
else if (t_counter>0)&&(double_value_of_time<testtime2))
{
    if (tran->tozone->z_id==zone_id)
    {
        cout<<"\nInteracted with "+"<<person_id<<" at "<<zone_id;
    }
    t_counter++;

    }
}
```