4-2003

User Interface Improvement for MLPQ System

Shasha Wu

University of Nebraska - Lincoln

Follow this and additional works at: https://digitalcommons.unl.edu/computerscidiss

Part of the Computer Engineering Commons, and the Computer Sciences Commons


https://digitalcommons.unl.edu/computerscidiss/142

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in Computer Science and Engineering: Theses, Dissertations, and Student Research by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.
User Interface Improvement for MLPQ System

by

Shasha Wu

A THESIS

Presented to the Faculty of
The Graduate College at the University of Nebraska
In Partial Fulfillment of Requirements
For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Peter Revesz

Lincoln, Nebraska
April 2003
User Interface Improvement for MLPQ System

Shasha Wu, M.S.
University of Nebraska, 2003

Advisor: Peter Revesz

This thesis describes the experience of migrating the MLPQ constraint database system, a complex standalone Multiple Document Interface (MDI) application, to a server-based remote accessible application. Centralized, standalone MDI application is a common style for personal software products in Windows. For a database management system, server-based, thin-client computing is a more popular infrastructure. Migrating an existing standalone constraint database application to be a web accessible constraint database server is the main goal of this thesis. This migration process provides a method for the constraint database system to collaborate with other specific applications.

We rebuild the desktop MLPQ constraint database system to be a web constraint database server. The analysis method, design model and sample codes in this thesis can be considered as a successful development sample for other development of migrating a desktop MDI system to a remote accessible web service application.

We also design and implement a function that finds the complement of a 2-D geometric constraint relation.
ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Peter Revesz, for his generous support, guidance and encouragement during the work of this thesis. I would also like to thank Professor Berthe Choueiry and Professor Vinodchandran Variyam for taking the time to serve on my defense committee. Professor Mohamed Fayad also deserves some thanks for teaching me how to design and document.

I would like to thank many of my friends at University of Nebraska-Lincoln. Special thanks to Hui Cheng for his help in implementing some of the functions used for this thesis and Scot Anderson for some valuable comments on this thesis. Finally, I would like to thank my wife Wanjuan for her love and care during the research on this thesis.
# Contents

**CHAPTER 1: INTRODUCTION**

1.1 INTRODUCTION ......................................................................................................................... 1  
1.2 BACKGROUND INFORMATION ................................................................................................. 1  

**CHAPTER 2: ANALYSIS** ............................................................................................................. 4  
2.1 MAKING MLPQ SYSTEM WEB ACCESSIBLE ............................................................................. 4  
2.2 POLICE EMERGENCY PROJECT REQUIREMENTS ...................................................................... 8  
2.3 NEGATION OPERATOR .............................................................................................................. 9  

**CHAPTER 3: DESIGN** ............................................................................................................... 11  
3.1 MLPQ WEB ACCESSIBLE SYSTEM DESIGN .......................................................................... 11  
3.2 POLICE EMERGENCY WEBSITE DESIGN ............................................................................. 16  
3.3 ALGORITHM FOR NEGATION OPERATOR ............................................................................... 19  

**CHAPTER 4: IMPLEMENTATION** ............................................................................................ 20  
4.1 MLPQ WEB ACCESSIBLE SERVER ........................................................................................... 20  
4.2 IMPLEMENTATION OF POLICE EMERGENCY WEBSITE ................................................ .. 23  
4.3 IMPLEMENTATION OF NEGATION OPERATOR IN MLPQ ..................................................... 32  

**CHAPTER 5: RESULT ANALYSIS** ........................................................................................... 35  
5.1 MLPQ WEB ACCESSIBLE SERVER ........................................................................................... 35  
5.2 POLICE EMERGENCY APPLICATION ...................................................................................... 36  
5.3 NEGATION OPERATOR ............................................................................................................ 37  

**CHAPTER 6: FUTURE WORK** .................................................................................................. 39  

**BIBLIOGRAPHY** ...................................................................................................................... 41  

**APPENDIX** .............................................................................................................................. 43  
APPENDIX A SOFTWARE PACKAGE LIST .................................................................................. 43  
APPENDIX B MLPQ WEB ACCESS SYSTEM USER’S INSTRUCTION ....................................... 44
Chapter 1: Introduction

1.1 Introduction

The multiple document interface (MDI) is a specification that defines the standard user interface for applications written for Windows. An MDI application enables a user to work with more than one document at a time. MDI is a popular interface because it allows opening multiple forms in one application. Examples of MDI applications include Microsoft Word, Microsoft Excel, and Microsoft PowerPoint. The original MLPQ constraint database system is a centralized, standalone desktop MDI application.

For a server-based database system, thin-client computing is becoming one of the most popular infrastructure in recent years. Migrating the existing MLPQ standalone desktop application to be a web accessible server is the main goal of this thesis. This migration process is a process to provide a method for the core constraint database system to collaborate with other real applications.

In this thesis, we rebuild the MLPQ constraint database system, a standalone MDI desktop application, to be a web accessible constraint database server. We hope that the analysis method and design model in this thesis can be considered as a successful development example for other similar projects.

1.2 Background information

In this thesis, we assume some familiarity with constraint database which can be found in [KKR95, Revesz02]. The basic idea in Constraint database is the introduction of constraint formulae as a basic data type in databases. The notion of constraint data relies on a simple and fundamental duality: a constraint (formula) $\phi$ in free variables $x_1, \ldots, x_n$
is interpreted as a set of tuples $(a_1, \ldots, a_n)$ over the schema $x_1, \ldots, x_n$ that satisfy $\phi$; and, conversely, a finitely representable object in $x_1, \ldots, x_n$ space can be viewed as a constraint [KKR90, Revesz02].

MLPQ is the abbreviation for Management of Linear Programming Queries. It is a constraint database system that implements rational linear constraint databases and queries. It allows SQL and Datalog queries, minimum and maximum aggregation operators over linear objective functions, and some other operators [Revesz02, p302]. The current version of the MLPQ/PReSTO system merges the MLPQ/GIS and the PReSTO constraint database systems, which were both built at the University of Nebraska-Lincoln. The executable file of the MLPQ/PReSTO system, running on Microsoft Windows 2000, and all the sample text files listed in the appendix are available free as down-loadable files from the webpage: http://www.cse.unl.edu/revesz, which also contains a list of publications related to both the MLPQ/GIS and the PReSTO systems. By using constraints which contain arithmetic predicates, we can express queries that are not supported in traditional query languages. More and more people become interested in applying MLPQ system in their own applications. With the increased number of users, the demand for improving its user-friendliness also increased. One way to improve user-friendliness is to enable web-based application development.

Currently, with the usage of Internet and mobile calculation becoming more and more popular, almost all industry database systems have the ability to support remote access. The MLPQ system in its origin is a centralized, standalone desktop MDI application. To make it remote accessible and web accessible is a big challenge for the system development. But without this capability, the usage of this system is limited. Furthermore,
through remote access methods, people can design their own user-interface in their applications. This is very helpful in partially solving the user interface problem.
Chapter 2: Analysis

2.1 Making MLPQ System Web Accessible

A desktop MDI application is a window that makes up the background of a multiple-document interface. It has three kinds of windows: a frame window, a client window, and a number of child windows. The frame window is like the main window of an application, and it surrounds the client window. The client window is a child of the frame window and serves as the background for the child windows. The client window also provides support for creating and manipulating the child windows in which documents are displayed. Each MDI application consists of one or more parent windows, each containing an MDI client area—the area where the child forms will be displayed. You can write code to display as many instances of the child forms as you want [MSDN]. It is a popular application format for personal application in Windows system. Figure 2.1.1 shows a basic MDI application in use.

![Basic MDI application](image)

*Figure 2.1.1 Basic MDI application [MSDN]*
As a result of the limitations of file sharing architectures, the client/server architecture emerged. The term client/server was first used in the 1980s in reference to personal computers on a network. The actual client/server model started gaining acceptance in the late 1980s. The client/server software architecture is a versatile, message-based and modular infrastructure that is intended to improve usability, flexibility, interoperability, and scalability as compared to centralized, mainframe, time sharing computing [Sadoski97]. The client/server architecture reduced network traffic by providing a query response rather than total file transfer. It improves multi-user updating through a GUI front end to a shared database [Schussel95, Edelstein94]. A client is defined as a requester of services and a server is defined as the provider of services. A single machine can be both a client and a server depending on the software configuration [Sadoski97].

The MLPQ web accessible server makes the original standalone desktop MLPQ application remote accessible and provides a set of interfaces for other applications to use the constraint database from client machines. So there are two major goals for this problem: providing a remote access schema and making sure it has the ability to collaborate with other applications. The detailed requirements we try to achieve can be illustrated as follows.

1. **Remote accessible.**

Remote accessible means the user can read and write to the database system through the network. The basic functions for this purpose include:

*Connecting and Login*: User can setup a connection between the database system and the client machine through the network. Login process and user id are required to specify different users. This is a basic requirement for a server-based database system.
Open: User can open a specific constraint database in their own application.

Query: User can send SQL or datalog queries to the server and have them executed on the server machine. After the execution, the server should return a text result to the requester through the network. This is a basic requirement for a database system.

Receive the graphic result: User can achieve the graphic result from the server machine. Graphic result is one of the most important features that MLPQ system can provide. The new MLPQ web accessible server application should have the same capability.

Close and Logout: The server can release all resources that were allocated to a user after the user exits. This is important for the server to run efficiently for a long time.

2. Easy to use.

The MLPQ web accessible constraint database system should act as a basis for the development of other specific applications. Programmer-friendly application programming interfaces (APIs) that provide for the rapid development of database oriented custom applications is one of the most important issues that encourage developers to implement this system. Therefore, the new MLPQ server should improve its usability and interoperability. Usability means “the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component” [IEEE90]. Interoperability means “the ability of two or more systems or components to exchange information and to use the information that has been exchanged” [IEEE90]. To achieve these goals, the interface of the system should be constant, concise and complete.

Keeping interfaces constant is very important for the MLPQ web accessible server because it is not a standalone program. There are lots of other applications dependent on it. Therefore, every change on interfaces may cause a lot of maintenance work beyond the
system itself. “Concise” means the interface should be simple and easy to apply.
“Complete” means the system should provide enough functionality for the common
requirements.

3. Keep consistency in MLPQ’s core functions.

During several years’ using and testing, the MLPQ/PReSTO system has proved its
correctness. The new MLPQ web access system must keep the core functions consistent.
This is the best way to ensure the correctness of the new program. If there are lots of
modifications on its original codes, it may generate a lot of work to make sure that we
can get the same result as the original program.

4. Flexibility.

Flexibility means “the ease with which a system or component can be modified for use in
applications or environments other than those for which it was specifically designed”
[IEEE90]. From the system developers’ view, this means the system should have the
ability to extend its functionality and the extension should be transparent to the user.
From application developers’ perspective, this means they are able to extend the usage of
the system in different domains easily.

5. Maintainability.

To the system developer, the design and implementation of the system should be easy to
read and understand. The structure of the system should be clear and organized. To
application developers, they can build a maintainable system only if its dependent system
is constant and complete. To build a maintainable application based on an unstable
system is as unpractical as to build a house on the sand.

6. Multi-users support.
A real web accessible application must have the ability to support multi-users to access the system at the same time. Different users can do their own work in their own context. The interference among different users must be limited to a very low level. Every user should feel that the server just works for himself alone at any time.

2.2 Police Emergency Project Requirements

The Police Emergency application is a sample implementation that applies the functions provided by the MLPQ web accessible server in a three-tier architecture. Suppose we know about a town represented by constraint database as follows.

**Emergency** (Type, No, Street, T) describes what type of emergency occurs at address (Number, Street) at time T.

**Resident** (Name, No, Street) describes which person (Name) resides at which address (Number, Street).

**Location** (No, Street, X, Y) describes which house (Number, Street) is at which location (X, Y).

**Contains** (Street, X, Y) describes which Street contains which locations (X, Y).

**Police** (Name, VIN, X, Y, T) describes which police officer (Name) drives car with vehicle identification number VIN at location (X, Y) and time T.

The functions we want to realize in the police emergency application include

1. Find and display a street upon the town map.
2. Find the position of the police officer at a given time.
3. Find the location of the emergencies during a given time interval.
4. Find the house location of the address.
5. Find the given resident’s house location.

6. Find the places where the given police officer is reachable at time $T$.

In the output of the query, the entire map of the town should be shown in black as the background, and the result of each query should be highlighted in red.

### 2.3 Negation Operator

The negation operator has the following syntax:

$$\overline{R(x_1, \cdots, x_n)} := \text{not } R(x_1, \cdots, x_n).$$

$R$ could be a relation with linear constraints over the relationals or polynomial constraints over the reals. $R$ could also be a relation with $=, \neq, <,$ and $\leq$ constraints over the integers.

In all of these cases it is possible to find a constraint representation of the complement of $R$. The constraint representation will contain the same type of constraints as the input relation. This is called a closed-form representation. [Peter 02, Page 59]

The negation operator was not available in the original MLPQ system. The reason is that a general negation operation requires extensive computation. For any given constraint relation $R$, we assume $p_1, p_2, \ldots, p_m$ are tuples of $R$. Every tuple consists of a set of atomic constraints as follows:

$$p_1 = r_{1,1}, r_{1,2}, \cdots, r_{1,k_1}.$$

$$p_2 = r_{2,1}, r_{2,2}, \cdots, r_{2,k_2}.$$

$$\cdots$$

$$p_m = r_{m,1}, r_{m,2}, \cdots, r_{m,k_m}.$$

Then we can represent $R$ and $\overline{R}$ as follows:
\[ R = p_1 \lor p_2 \lor \cdots \lor p_n \]
\[ = (r_{1,1} \land r_{1,2} \land \cdots \land r_{1,k_1}) \lor (r_{2,1} \land r_{2,2} \land \cdots \land r_{2,k_2}) \lor \cdots \lor (r_{m,1} \land r_{m,2} \land \cdots \land r_{m,k_m}) \]

\[
\bar{R} = p_1 \lor \cdots \lor p_{m-1} \lor p_m = p_1 \land \cdots \land p_{m-1} \land p_m
\]
\[
= (r_{1,1} \land \cdots \land r_{1,k_1} \land \bar{r}_{1,1}) \land (r_{2,1} \land \cdots \land r_{2,k_2} \land \bar{r}_{2,1}) \land \cdots \land (r_{m,1} \land \cdots \land r_{m,k_m} \land \bar{r}_{m,1})
\]
\[
= (r_{1,1} \lor \cdots \lor r_{1,k_1-1} \lor r_{1,k_1}) \land (r_{2,1} \lor \cdots \lor r_{2,k_2-1} \lor r_{2,k_2}) \land \cdots \land (r_{m,1} \lor \cdots \lor r_{m,k_m-1} \lor r_{m,k_m})
\]
\[
= \bigcup_{i=1}^{m} r_{i,j}
\]
\[
= (r_{1,1} \land \cdots \land r_{m-1,1} \land \bar{r}_{m,1}) \lor (r_{1,1} \land \cdots \land r_{m-1,1} \land \bar{r}_{m,1}) \lor \cdots \lor (r_{1,1} \land \cdots \land r_{m-1,1} \land \bar{r}_{m,1})
\]
\[
= (r_{1,1} \land \cdots \land r_{m-1,2} \land \bar{r}_{m,1}) \lor (r_{1,1} \land \cdots \land r_{m-1,2} \land \bar{r}_{m,1}) \lor \cdots \lor (r_{1,1} \land \cdots \land r_{m-1,2} \land \bar{r}_{m,1})
\]
\[
= \cdots
\]
\[
= (r_{1,k_1} \land \cdots \land r_{m-1,k_{m-1}} \land \bar{r}_{m,1}) \lor (r_{1,k_1} \land \cdots \land r_{m-1,k_{m-1}} \land \bar{r}_{m,1}) \lor \cdots \lor (r_{1,k_1} \land \cdots \land r_{m-1,k_{m-1}} \land \bar{r}_{m,1})
\]

Assume \( m \) is the number of tuples in the relation \( R \) and \( k \) is the maximum number of atomic constraints in any tuple of \( R \). It is easy to see that:

\( R \) has \( m \) tuples. Each tuple has at most \( k \) atomic constraints. The space complexity of \( R \) is:

\[ \sum_{i=1}^{m} k_i = O(mk) \]

\( \bar{R} \) has \( \prod_{i=1}^{m} k_i \leq \prod_{i=1}^{m} k = k^m \) tuples. Every tuple has \( m \) atomic constraints. Therefore, the space complexity of \( \bar{R} \) is:

\[ m \prod_{i=1}^{m} k_i = O(mk^m) \]
Chapter 3: Design

3.1 MLPQ Web Accessible System Design

The MLPQ web accessible server and the dependent applications are collaborative systems. To describe the whole infrastructure of the system more clearly, we introduce the three-tier software architecture. The three-tier software architecture (also called three layer architecture) emerged in the 1990s to overcome the limitations of the two-tier architecture [SC97]. The third tier (middle tier) is between the user interface (client) and the data management (server) components. This middle tier provides process management where business logic and rules are executed. The three-tier architecture is used when an effective distributed client/server design is needed that provides increased performance, flexibility, maintainability, reusability, and scalability, while hiding the complexity of distributed processing from the user. [SC97]

As we showed in Figure 3.1.1, the MLPQ web accessible programs can be grouped into three tiers. Each tier has its own responsibilities that are listed in Table 3.1.

![Figure 3.1.1 MLPQ Web Access System (B/S) Infrastructures](image)
The first tier is the MLPQ Constraint Database Server. The server program uses socket to communicate with the components in middle tier. Socket is one of the most fundamental technologies of computer networking. It has been in widespread use since the early 1980s. It is famous for its portability, language-independency, and relative simplicity. Socket is an endpoint of communication. It represents a single connection between two network applications [SGG02]. These two applications nominally run on different computers, but sockets can also be used for interprocess communication on a single computer. Through socket, applications can communicate with each other using standard mechanisms already built into network hardware and operating systems. It works across all operating systems and hardware that support TCP/IP.

The second tier is middleware layer. It is a set of components that work between the HTML files and the MLPQ web accessible server. These components act like a kind of
“glue” that combines the user’s web interfaces and the MLPQ web accessible server. To reach this goal, the programs in this tier should be able to communicate with the browser and the MLPQ server through TCP/IP and socket. Since TCP/IP and socket are all standard communication mechanisms, people have many technique choices to realize them. As we showed in figure 3.1.1, ASP, Java Applet, or CGI are all possible approaches for those components.

The third tier is the user interface (client). It is common to use HTML language in constructing user interfaces for the web application.

Figure 3.1.1 describes a Browser/Server (B/S) system. In a B/S structure, the client is a “thin” client. There are few performance requirements for the client machine. Thin clients are small, affordable, easy to maintain, reliable, and secure. Server-based, thin-client computing offers unparalleled flexibility to foster collaboration, exploration, and discovery [Mitchell02]. Client user can avoid installing any unfamiliar software into their personal computer in order to run the application. The only software they must have is an Internet browser. This is the most popular infrastructure for web application. So we built our sample application (police emergency application) based on this architecture.

However, two-tier architecture is also supported by the MLPQ web accessible server. It is also called fat-client software architecture because users have to install and configure client software on their personal computer. The client software is used to provide the user interfaces, do calculations and communicate with the database server. The benefit of this architecture is that the client machine can partially share the workload with the server machine and it has the ability to provide more complicate graphic user-interface (GUI) controls for the client user. We also provide a sample program for this kind of application.
This program provides a new user interface for Datalog and SQL queries based on the MLPQ web accessible server. The two-tier architecture is displayed in Figure 3.1.2. Figure 3.1.3 is the user interfaces of the client program.

Figure 3.1.2 MLPQ Web Access System (C/S) Infrastructures

Figure 3.1.3 Interfaces of client.exe based on C/S Infrastructures
Object-Oriented programming is the most popular software developing method. The Unified Modeling Language (UML) is one of the most popular tools for Object-Oriented design and modeling [BRJ98]. We can use class diagram based on UML to show the detail design of the MLPQ web accessible server. Figure 3.1.3 is part of the Class Diagram for the new MLPQ web accessible server.

The source code of the whole MLPQ program has over 200 files and 135 classes. It is impossible to draw a complete class diagram here. Figure 3.1.3 only displays the classes that relate to our remote accessing functions. CNewSocket is the most important class in this diagram. It is a newly created class inherited from CAsyncSocket. When the MLPQ
server is running, it will listen to the predefined socket port and wait to receive connections and input messages. When the client application sends a string command to the socket port, the MLPQ server will automatically pick the command out and parse it based on a command list. According to the parsing result, the CNewSocket can activate related functions in DB, CIdbDoc and CgraphView to fulfill the request.

### 3.2 Police Emergency Website Design

To satisfy the requirements proposed in 2.2, we designed eight functions for this application. The definition and solution for each function is listed below.

1. **Login:** Login page will try to connect to MLPQ server and initialize all session variables. After that, it will redirect the user to the main page.

2. **Town Map:** This function is to display the street of the town. It accepts a street name and returns the whole map of the town with the requested street in red color. The SQL query is:

   ```sql
   SELECT Contains.Street, Contains.x, Contains.y
   FROM Contains
   WHERE Contains.Street = inputStreet
   ```

3. **Officer:** This function finds the position of the police officer at a given time. It accepts the police officer’s name and a time instance. The result is the whole map of the town with the officer’s position in red color. The SQL query is:

   ```sql
   SELECT Police.Name, Police.x, Police.y
   ```
FROM Police
WHERE Police.Name = policename, Police.T = time

4. **Emergency**: This function finds the location of the emergencies during the given time interval. It accepts a time interval and returns the whole map of the town with the locations of the emergencies that occur during the time interval in red color.

The SQL query is:

```
SELECT Emergency.Type, Location.x, Location.y
FROM Emergency, Location
WHERE Emergency.No = Location.No,
    Emergency.Street = Location.Street,
    Emergency.T >= starttime,
    Emergency.T <= endtime
```

5. **Locate**: This function finds the house at the given address. It accepts the address and returns the whole map of the town with the address location in red color.

The SQL query is:

```
SELECT Location.No, Location.x, Location.y
FROM Location
WHERE Location.No = No,
    Location.Street = Street
```

If the user does not know the exact address, it allows partial matching in the query. For example, if the user can not remember the street name, this page can return all houses whose number is the given value. Also, if the user doesn’t know exactly the number of the address, this page can return all the locations of the houses located in a given street.
6. **Resident**: This function finds the house of the given resident. It accepts the name of the resident and returns the whole map of the town with the resident’s address location in red color.

The SQL query is:

```sql
SELECT Location.No, Location.x, Location.y
FROM Location, Resident
WHERE Location.No = Resident.No,
      Location.Street = Resident.Street,
      Resident.Name = name
```

7. **Reachable**: This function finds the places where the given police officer is reachable from his location at time $T$. It accepts the name of the police officer and the time. The result is the whole map of the town with the officer’s reachable area in red color.

The Datalog is

```
Reach(n) :- Contains(n, x, y), Police(Name, VIN, x, y, T),
          Name = pname, T = time.

Reach(n) :- Reach(m), Contains(m, x, y), Contains(n, x, y).

Reachable(n, x, y) :- Reach(n), Contains(n, x, y).
```

8. **Logout**: The logout function will connect to the MLPQ system and release all resources related to the current user. After that, it will initialize the session variables and redirect the user to the login page.
3.3 Algorithm for the Negation Operator

Based on the analysis in Chapter 2.3, we can implement the negation operator on 2-dimensional geometric constraint relations at the form $R(id, x, y)$. For a given relation $R$, we have:

$$R(id, x, y) = p_1 \lor p_2 \lor \cdots \lor p_m$$

$$= (r_{i,1} \land r_{i,2} \land \cdots \land r_{i,k_i}) \lor (r_{2,1} \land r_{2,2} \land \cdots \land r_{2,k_2}) \lor \cdots \lor (r_{m,1} \land r_{m,2} \land \cdots \land r_{m,k_m})$$

$$\overline{R(id, x, y)} = \bigcap_{i=1}^{m} \left( \bigcup_{j=1}^{k_i} \overline{r_{i,j}} \right)$$

$$= (\overline{r_{i,1}} \land \cdots \land \overline{r_{m-1,1}} \land \overline{r_{m,1}}) \lor (\overline{r_{i,1}} \land \cdots \land \overline{r_{m-1,2}} \land \overline{r_{m,2}}) \lor \cdots \lor (\overline{r_{i,1}} \land \cdots \land \overline{r_{m-1,k_i}} \land \overline{r_{m,k_i}})$$

$$\lor \cdots$$

$$\lor (\overline{r_{i,k_i}} \land \cdots \land \overline{r_{m-1,k_i-1}} \land \overline{r_{m,k_i-1}}) \lor (\overline{r_{i,k_i}} \land \cdots \land \overline{r_{m-1,k_i}} \land \overline{r_{m,k_i}}) \lor \cdots \lor (\overline{r_{i,k_i}} \land \cdots \land \overline{r_{m-1,k_m-1}} \land \overline{r_{m,k_m}})$$

All $r_{i,j}$ in the formula are atomic constraints that contain either $x$ or $y$. The algorithm can be displayed as follows:

Assume relation $R$ has $m$ tuples, count($t_i$) return the number of atomic constraints in $t_i$. The new complement relation is stored in $nR$.

```plaintext
nR = new(Relation)  //Create a new relation
for i1=1 to count(R.t1)
    for i2=1 to count(R.t2)
        ...
        for in=1 to count(R.tn)  //add a new tuple t to nR
            t = nR.new(tuple)  //add an atomic constraint to tuple t
            for k=1 to m
                add(Complement(R.t.k, c_k), t)
            end of loop k
        end of loop in
    end of loop i2
end of loop i1
```
Chapter 4: Implementation

4.1 MLPQ Web Accessible Server

Table 4.1 lists all commands and their arguments recognized by the MLPQ web accessible server. <user> records the login user name. <filename> is the name of constraint database opened in the server. <relation name> provides the name of the new relation that will be created on the MLPQ web accessible server. <color> is used to assign a color for the selected relation. Table 4.1.2 maps the value of <color> to the actual color assigned to the relation. The execution of each command may return 0, 1, or a file name. “1” means there are some errors within the execution. “0” result tells people that the command is executed successfully. If a file name was returned, it also means the command is successfully executed.

<table>
<thead>
<tr>
<th>Command</th>
<th>Arguments</th>
<th>Actions on CDB</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open</td>
<td>&lt;user&gt; &lt;filename&gt;$</td>
<td>Create a view for the user and open a CDB data file.</td>
<td>0 (success)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 (fail)</td>
</tr>
<tr>
<td>Close</td>
<td>&lt;user&gt; &lt;filename&gt;$</td>
<td>Close the view of the user.</td>
<td>0 (success)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 (fail)</td>
</tr>
<tr>
<td>SQLBasic</td>
<td>&lt;user&gt; &lt;filename&gt;</td>
<td>Execute a SQL query on the user’s view.</td>
<td>File.txt#</td>
</tr>
<tr>
<td></td>
<td>&lt;relation name&gt;</td>
<td></td>
<td>1 (fail)</td>
</tr>
<tr>
<td></td>
<td>#&lt;select&gt;#&lt;from&gt;#&lt;where&gt;$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQLAggregate</td>
<td>&lt;user&gt; &lt;filename&gt;</td>
<td>Execute a SQL query on the user’s view.</td>
<td>File.txt#</td>
</tr>
<tr>
<td></td>
<td>&lt;relation name&gt;</td>
<td></td>
<td>1 (fail)</td>
</tr>
<tr>
<td></td>
<td>#&lt;select&gt;#&lt;from&gt;#&lt;where&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#&lt;group&gt;#&lt;having&gt;$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQLSet</td>
<td>&lt;user&gt; &lt;filename&gt;</td>
<td>Execute a SQL query on the user’s view.</td>
<td>File.txt#</td>
</tr>
<tr>
<td></td>
<td>&lt; relation name&gt;</td>
<td></td>
<td>1 (fail)</td>
</tr>
<tr>
<td></td>
<td>#&lt;select&gt;#&lt;from&gt;#&lt;where&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#&lt;group&gt;#&lt;having&gt;$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# SQL Nested

Execute a SQL query on the user’s view.

<table>
<thead>
<tr>
<th>SQLNested</th>
<th>&lt;user&gt; &lt;filename&gt; &lt;relation name&gt;</th>
<th>Execute a SQL query on the user’s view.</th>
<th>File.txt# 1 (fail)</th>
</tr>
</thead>
</table>

## Datalog

Execute a Datalog query on the user’s view.

<table>
<thead>
<tr>
<th>Datalog</th>
<th>&lt;user&gt; &lt;filename&gt; &lt;datalog string&gt;</th>
<th>Execute a Datalog query on the user’s view.</th>
<th>file.txt# 1 (fail)</th>
</tr>
</thead>
</table>

## Include

Highlight the relation by its name and assign a color for it in the view.

<table>
<thead>
<tr>
<th>Include</th>
<th>&lt;user&gt; &lt;filename&gt; &lt;relation name&gt; &lt;color&gt;</th>
<th>Highlight the relation by its name and assign a color for it in the view.</th>
<th>0 (success) 1 (fail)</th>
</tr>
</thead>
</table>

## Clear

Deselect all relations in the view.

<table>
<thead>
<tr>
<th>Clear</th>
<th>&lt;user&gt; &lt;filename&gt;</th>
<th>Deselect all relations in the view</th>
<th>0 (success) 1 (fail)</th>
</tr>
</thead>
</table>

## Get Image

Copy the image of the view in screen and save to disk.

<table>
<thead>
<tr>
<th>GetImage</th>
<th>&lt;user&gt; &lt;filename&gt;</th>
<th>Copy the image of the view in screen and save to disk.</th>
<th>File.bmp# 1: fail</th>
</tr>
</thead>
</table>

### Table 4.1.1 Command list to MLPQ Web Access Program

<table>
<thead>
<tr>
<th>Value of &lt;color&gt;</th>
<th>Color of the relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>1</td>
<td>Red</td>
</tr>
<tr>
<td>2</td>
<td>Green</td>
</tr>
<tr>
<td>3</td>
<td>Blue</td>
</tr>
</tbody>
</table>

### Table 4.1.2 Map from the value of <color> to actual color of the relation
For better understanding how the command works, we provide a sample execution sequence as follows.

```
Open sam police.txt$

SQLBasic sam police.txt vine#Contains.Street, Contains.x, Contains.y#Contains#
   Contains.Street= 'Vine'$
   // SELECT Contains.Street, Contains.x, Contains.y FROM Contains WHERE Contains.Street= 'Vine'
Datalog sam police.txt oak(id, x, y):- Contains(id, x, y), id='Oak'.$
Include sam police.txt Contains 0$    //Show the street in black color
Include sam police.txt oak 2$          //Show oak in green color
Include sam police.txt vine 1$         //Show vine in red color
GetImage sam police.txt$                //Return the image
Close sam police.txt$                   //Release resources in the server
```

* sam is a user name, police.txt is a constraint database file. Contains is a relation in police.txt.

Figure 4.1.1 is the main window of MLPQ web accessible server. To make it work for web access, the “listen” menu should be selected first. The default environment settings are saved in mlpq_nt.cfg file and the values of the settings are displayed in Table 4.1.3.

The system administrator can modify these settings in the file to configure the server.

<table>
<thead>
<tr>
<th>Name and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WorkDir = “c:\inetpub\wwwroot\dbs\”</td>
<td>Define the directory to load constraint database files by the server.</td>
</tr>
<tr>
<td>OutPut = “c:\inetpub\wwwroot\dbs\”</td>
<td>Define the directory for the server to output its result file.</td>
</tr>
<tr>
<td>Port = “2222”</td>
<td>Define the socket port value</td>
</tr>
</tbody>
</table>

*Table 4.1.3 Settings in mlpq_nt.cfg*
4.2 Implementation of Police Emergency Website

Login: The login.html accepts the user’s name and sends it to login.asp. Login.asp will save the string to a session variable “USR”. There are three session variables defined in Login.asp. All of them will be used in all the queries before a user finally logs out. The variable “SERVER” is used to store the server’s IP address. “USR” stores the user’s login id. “VIEW_NM” keeps the number of queries that have been executed by this user. The last variable is very useful to make sure that every new relation we generated in the server has a unique name. Figure 4.2.1 shows the login page. After the login process, it will go to the home page of the Police Emergency Website shown in Figure 4.2.2. From the home page, the user can use the buttons in the bottom to do the queries.

```
<% Session.Contents("SERVER") = Request.ServerVariables("SERVER_NAME") Session.Contents("USR") = Request("STR_USR") Session.Contents("VIEW_NM") = 0 Response.Redirect "a1.htm" %>
```
Figure 4.2.1 Login page for the Police Emergency Website

Figure 4.2.2 Home page of the Police Emergency Website
Below are four sample code segments used in the ASP file. The first one is to create a `Socket.TCP` object and connect it to the server socket. The second one is to open a database file in the server. The third is to select the display relations and the last one is to return the image. The codes here and after are all written by VBScript and ASP.

<table>
<thead>
<tr>
<th>Create a Socket.TCP object and connect to the server:</th>
</tr>
</thead>
<tbody>
<tr>
<td>asObj = Server.CreateObject( &quot;Socket.TCP&quot; )</td>
</tr>
<tr>
<td>asObj.Host = Session.Contents(&quot;SERVER&quot;) &amp; &quot;:2222&quot;</td>
</tr>
<tr>
<td>asObj.Open</td>
</tr>
<tr>
<td>asObj.Wait</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Open a database file:</th>
</tr>
</thead>
<tbody>
<tr>
<td>str = &quot;Open &quot; &amp; Session.Contents(&quot;USR&quot;) &amp; &quot; police.txt$&quot;</td>
</tr>
<tr>
<td>asObj.SendText str</td>
</tr>
<tr>
<td>asObj.Wait</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Select a relation:</th>
</tr>
</thead>
<tbody>
<tr>
<td>str = &quot;Include &quot; &amp; Session.Contents(&quot;USR&quot;) &amp; &quot; police.txt Townmap 0$&quot;</td>
</tr>
<tr>
<td>asObj.SendText str</td>
</tr>
<tr>
<td>asObj.Wait</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return the image:</th>
</tr>
</thead>
<tbody>
<tr>
<td>str = &quot;GetImage &quot; &amp; Session.Contents(&quot;USR&quot;) &amp; &quot; police.txt$&quot;</td>
</tr>
<tr>
<td>asObj.SendText str</td>
</tr>
<tr>
<td>asObj.Wait</td>
</tr>
</tbody>
</table>
Figure 4.2.3 Town map query and its result page

Figure 4.2.3 (a) shows the result of clicking the “Town Map” button. This page has an input edit box for the user to input a street name. Figure 4.2.3(b) shows the result of executing the query to find “Vine” street. The core ASP code for 4.2.3 is shown below:

```sql
SQL query in ASP file to find the street:
Session.Contents("VIEW_NM") = Session.Contents("VIEW_NM") + 1
str = "SQLBasic " & Session.Contents("USR") & " police.txt " & Session.Contents("USR")
   & Session.Contents("VIEW_NM")
str = str & "# Contains.Street, Contains.x, Contains.y"
str = str & "# Contains"
if Request("STR_STREET") = "" Then
   str = str & "$"
Else str = str & "# Contains.Street= " & Request("STR_STREET") & "$"
End if
asObj.SendText str
asObj.Wait
```
Figure 4.2.4 Page to find the location of Police Officer and the result

Figure 4.2.4 (a) shows the result of clicking the “Officer” button. It has a dropdown list to select the officer and an edit box for the time. Figure 4.2.4(b) shows the result of executing the query to find “Charles” at time 12. The core ASP code for 4.2.4 is shown below.

**SQL query in ASP file to find the location of the police officer:**

```sql
Session.Contents("VIEW_NM") = Session.Contents("VIEW_NM") + 1
str = "SQLBasic " & Session.Contents("USR") & " police.txt " & Session.Contents("USR") & Session.Contents("VIEW_NM")
str = str & "# Police.Name, Police.x, Police.y"
str = str & "# Police"
str = str & "# Police.Name= " & Request("STR_NAME")
str = str & ". Police.T=" & Request("STR_TIME")
str = str & "$
asObj.SendText
str asObj.Wait
```
Figure 4.2.5 Page to find the Emergencies and its result

Figure 4.2.5 (a) shows the result of clicking the “Emergency” button. It has two edit boxes for the time interval. Figure 4.2.5(b) shows the result of executing the query to find all emergencies that occurred between time 10 and 40. The core ASP code for 4.2.5 is shown below

```
SQL query in ASP file to find emergencies:
Session.Contents("VIEW_NM") = Session.Contents("VIEW_NM") + 1
str = "SQLBasic " & Session.Contents("USR") & " police.txt " & Session.Contents("USR") & 
    Session.Contents("VIEW_NM")
str = str & "; Emergency.Type, Location.x, Location.y"
str = str & "; Emergency, Location"
str = str & "; Emergency.No=Location.No, Emergency.Street=Location.Street"
str = str & ", Emergency.T>= " & Request("STR_TIME1")
str = str & ", Emergency.T<="& Request("STR_TIME2")
str = str & ";$"
asObj.SendText
str asObj.Wait
```
Figure 4.2.6 shows the result of clicking the “Locate” button. It has two edit boxes for the address number and street. Figure 4.2.6 (b) shows the result of executing the query to find the location of “1000 Vine”. The ASP code for 4.2.6 is shown below.

**SQL query in ASP file to find the location of the house:**
```
Session.Contents("VIEW_NM") = Session.Contents("VIEW_NM") + 1
str = "SQLBasic " & Session.Contents("USR") & " police.txt " & Session.Contents("USR") & Session.Contents("VIEW_NM")
str = str & "# Location.No, Location.x, Location.y"
str = str & "# Location"
If Request("STR_STREET") = "" Then str1 = ""
Else str1 = "Location.Street= " & Request("STR_STREET") & ""
End if
If Request("STR_NO") = "" Then str2 = ""
Else str2 = "Location.No= " & Request("STR_NO") & ""
End if
if (str1="") and (str2="") then str = str & ";"
extif str1="" then str = str & " " & str2 & ""
extif str2="" then str = str & " " & str1 & ""
ext else str = str & " " & str1 & ", " & str2 & ""
End if
asObj.SendText
str asObj.Wait
```
Figure 4.2.7 Page to find the Location of a Resident and its result

Figure 4.2.7 (a) shows the result of clicking the “Resident” button. It has a dropdown list to select the resident’s name. Figure 4.2.7 (b) shows the result of executing the query to find the house of “Carl”. The ASP code for 4.2.7 is shown below

```sql
SQL query in ASP file to find the house of the resident:
Session.Contents("VIEW_NM") = Session.Contents("VIEW_NM") + 1
str = "SQLBasic " & Session.Contents("USR") & " police.txt " & Session.Contents("USR") & " VIEW_NM"
str = str & ", Location.No=" & Location.No & ", Location.x=" & Location.x & ", Location.y=" & Location.y & ", Resident.Name=" & Resident.Name ""
asObj.SendText
str asObj.Wait
```
Figure 4.2.8 Page to find the reachable street for a given Police officer and its result

Figure 4.2.8 (a) shows the result of clicking the “Reachable” button. It has a dropdown list to select the officer’s name and an edit box for the time. Figure 4.2.8 (b) shows the result of executing the query to find the reachable streets of “Charles” at time 12. The ASP code for 4.2.8 is shown below.

**SQL query in ASP file to find the reachable street for the officer:**

```sql
Session.Contents("VIEW_NM") = Session.Contents("VIEW_NM") + 1
prefix = Session.Contents("USR") & Session.Contents("VIEW_NM")
str = "Datalog " & Session.Contents("USR") & " police.txt " & prefix
str = str & "Reach(n) :- Contains2(n,x,y), Police(Name, VIN, x, y, T), Name=" & Chr(34) & Request("STR_NAME") & Chr(34) & ", T=" & Request("STR_TIME") & ".$"
asObj.SendText str
asObj.Wait
str = "Datalog " & Session.Contents("USR") & " police.txt " & prefix
str = str & "Reach(n) :- & prefix & Reach(m), Contains2(m, x, y), Contains2(n, x, y).$"
asObj.SendText str
asObj.Wait
str = "Datalog " & Session.Contents("USR") & " police.txt " & prefix
str = str & "Rable(n, x, y) :- & prefix & Reach(n), Contains2(n, x, y).$"
asObj.SendText str
str asObj.Wait
```
The “Logout” button is used to close the view in MLPQ server and release related resources. After that, it will initialize all session variables and redirect the user to the login form.

```vbscript
asObj = Server.CreateObject( "Socket.TCP" )
asObj.TimeOut = 2000
asObj.Host = Session.Contents("SERVER") & ":2222"
asObj.Open
asObj.Wait
If asObj.Connected then
    str = "Close " & Session.Contents("USR") & " police.txt$
    asObj.SendText str
    asObj.Wait
    asObj.Close
End if
Session.Contents("SERVER") = ""
Session.Contents("USR") = ""
Session.Contents("VIEW_NM") = 0
Response.Redirect "Login.html"
```

4.3 Implementation of Negation Operator in MLPQ

To implement the negation function, we make the complement icon in MLPQ available when the user selects a 2-dimensional geometric relation. It will ask user to input a new relation name for the negation result. Then it will do the calculation on the selected relation and save the result to the new relation.

To apply the negation algorithm described in section 3.3, we use a recursive function as follows:

```c
CString cs[]; //Original Constraints
int tp[]; //index of first constraint in cs for each tuple
int cst[]; //index of constraints in result stack

CString DoAtomicComplement(CString cs)
{
    int pos;
    if (cs.Replace("="","\")==0)
        if (cs.Replace("","\")==0)
            if (cs.Replace("\","\")==0)
                if ($(pos=cs.Find('!'))==0)
                    cs = cs.Replace('!','>') + "+" + cs.Replace('!','<');
```
void RecurNegation(int tp_order, FILE * fp_out)
{
    if (tp_order is the end of tuples) {
        writeout(cs, cst, tp_order, fp_out);
    }
    else {
        // recursive do negation
        for(int i=tp[tp_order]; i<tp[tp_order+1]; i++){
            cst[tp_order] = i;  // pick one constraint from tuple tp_order.
            RecurNegation(tp_no+1, fp_out); //go to next tuple
        }
    }
}

void main()
{
    ...
    DoAtomicComplement(cs);
    RecurNegation(0, fp)
    ...

Figure 4.3.1 shows a sample 2-D geometric relation and its negation operation result.

(a) Original relation “rect2”
(b) Complement of "rect2"

Figure 4.3.1 Negation operation in MLPQ
Chapter 5: Result Analysis

5.1 MLPQ Web Accessible Server

We can analyze our design and implementation result based on the following issues discussed in Chapter 2:

Remote accessible. The sample application shows that the system really provides remote accessibility. The client program can work on the MLPQ server system through the network.

Easy to use. After we finish the sample police emergency application, we present it in course CSCE413/813 and ask students in the course to do their application assignment 2 based on this system. The response from the students is very encouraging. Most of the students can construct their web applications based on this system. When they achieve some knowledge of Active Server Page (ASP), VBScript programming language, and Internet Information Server (IIS) configuration, they can implement their designs in just several days by modifying the sample application. It shows that this system is quite easy to apply.

Keep consistency in MLPQ’s core functions. By providing a socket interface and a set of standard commands to activate original functions, the new MLPQ web accessible server keeps its original core functions untouched. This guarantees that the new MLPQ server can act the same as its original program.

Flexibility. Socket techniques are an industry standard today. Based on socket technique, The MLPQ web accessible server can be accessed through all operating systems and hardware that support TCP/IP. By increasing the number of its commands in the command list, system developers can improve its functions without disturbing the
existing applications and users. At the same time, application developers can easily extend the usage of the system in different domains. Based on socket techniques, the new MLPQ web accessible server is very flexible.

**Maintainability.** The new design and implementation of the system is based on Object-Oriented Programming and UML notations. It is easy to read and understand. The three-tier software architecture of the system is clear and organized.

**Multi-user support.** The socket is a multi-thread object. It can support multi-users natively. The MDI desktop program is also very help for this purpose. In the program, every user is assigned a MDI child form. Each MDI child form is an independent context for a user. Different users can do their work in their own context. The interference among different users is limited to a very low level. The conflict only happens when the number of users exceeds the system capability. In most situations, every user feels that the server just works for himself alone.

### 5.2 Police Emergency Application

An attractive and easy to understand user interface design is important. The web accessible police emergency application gives several good examples. It contains static and moving pictures related to the subject, an attractive layout of the available menu items. Each of the menu items is explained to the user, and flexibility is achieved by providing several choosable parameters for the user. Drop-down list of names avoids misspellings and facilitates the correctness of the queries. The sample code is easy to read and understand. Experience in the CSCE413/813 course shows that it is a good example for students to implement their applications.
5.3 Negation Operator

As we found, the negation function is a time consuming process. For a simple input relation, it will generate much more constraints and tuples than its original one. In our test, we defined a simple relation as follows

\[
\text{origin}(i,x,y): \begin{align*}
-i &= 1, \\
-x &> 0, \\
-x &< 7,
\end{align*}
\]

\[
\text{origin}(i,x,y): \begin{align*}
-i &= 2, \\
x &> 7, \\
x &< 8, \\
y &> 0, \\
y &< 7.
\end{align*}
\]

After the negation operation on this it, we get a list of tuples as follows

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
-x &> 0, \\
-x &< 7.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
-x &> 0, \\
x &> 8.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
x &> 8, \\
y &> 0.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
x &> 5, \\
-x &> 0.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
x &> 5, \\
y &> 0.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
x &> 5, \\
-x &> 0.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
x &> 5, \\
y &> 7.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
x &> 5, \\
-x &> 0.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
x &> 5, \\
y &> 7.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
x &> 8, \\
y &> 0.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
x &> 8, \\
y &> 0.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
y &> 7, \\
x &> 8.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
y &> 7, \\
-x &> 0.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
y &> 7, \\
y &> 0.
\end{align*}
\]

\[
\text{negation} (i, x, y) : \begin{align*}
-i &= 1, \\
y &> 7, \\
x &> 8.
\end{align*}
\]

There are 16 tuples in the result relation, which is 8 times to the original one. This calculation does not count the atomic constraints that only contain variable \(i\) in original relation. For example, original atomic constraint “\(i = 2\)” does not join to the calculation process and can not be found in the result. Variable \(i\) is arbitrarily assigned a value in each new tuple generated by the negation operation. This assignment is just to make the result relation visible in MLPQ draw graph function. Based on this assumption, the relation \(\text{origin}(i, x, y)\) has 2 tuples. Each tuple has 4 atomic constraints that are used in negation operation. According to Chapter 2.3, we have \(m=2, k=4\). Therefore, the result
relation should have $k^m = 4^2$ tuples. Each tuple has $m=2$ atomic constraints, except the one only containing variable $i$.

Removing the tuples that can’t be satisfied from the result relation, we can finally get a relation with 13 tuples in MLPQ system as follows:

```
negation (i, x, y) :- i=1, -x > 0, -x > -7.
negation (i, x, y) :- i=1, -x > 0, -y > 0.
negation (i, x, y) :- i=1, -x > 0, y > 7.
negation (i, x, y) :- i=1, x > 5, -x > -7.
negation (i, x, y) :- i=1, x > 5, x > 8.
negation (i, x, y) :- i=1, x > 5, -y > 0.
negation (i, x, y) :- i=1, x > 5, y > 7.
negation (i, x, y) :- i=1, -y > 0, -x > -7.
negation (i, x, y) :- i=1, -y > 0, x > 8.
negation (i, x, y) :- i=1, -y > 0, -y > 0.
negation (i, x, y) :- i=1, y > 7, -x > -7.
negation (i, x, y) :- i=1, y > 7, x > 8.
negation (i, x, y) :- i=1, y > 7, y > 7.
```

There are still some redundant tuples in the result. For example:

“negation (i, x, y) :- -x > 0, -x > -7.” is the same as “negation (i, x, y) :- -x > 0.”

“negation (i, x, y) :- x > 5, x > 8.” is the same as “negation (i, x, y) :- x > 8.”

“negation (i, x, y) :- y > 7, y > 7.” is the same as “negation (i, x, y) :- y > 7.”

The area described by the union of “negation (i, x, y) :- i=1, x > 5, x > 8.”, “negation (i, x, y) :- i=1, y > 7, x > 8.”, and “negation (i, x, y) :- i=1, -y > 0, x > 8.” can be expressed by one tuple “negation(i, x, y) :- i=1, x > 8.”.

To make the result more concise, we need a more efficient simplification function, which is already scheduled to realize in our next research topic.
Chapter 6: Future Work

In the MLPQ, there is no simplification process for the evaluation result. That not only makes the text result hard to understand, it also increases the complexity for the following evaluation. There are many simple or complicate rules that can be applied to simplify the constraint result. Applying those simplification rules may efficiently reduce the number of tuples in the result relation. As we discussed in the section 5.3, without the simplification process, the negation operation will generate many redundant tuples in the result. The simplification function is scheduled to be implemented in our next improvement.

In this MLPQ web accessible server, the login process is only used to generate a unique identification string for the server. We leave the security problem to the middleware and server platform. But in the real world, it is important that the database server has a user authorization and verification schema. [LM03] already discussed the possibility of applying datalog and constraint database to solve the authorization and access control problems in distributed systems. We plan to implement this idea in MLPQ web accessible server in the future.

We assume the server always has enough resources for every requester. However, this is not always true in a real world. Therefore, it is important to be able to set the maximum number of users for the server program based on the available resources. If the number of the active users exceeds the server’s capability, the system should be able to suspend new user’s login request.

As we described in the design chapter, people can access the MLPQ server through ASP, Java Applet, or CGI, etc. Providing samples for each technique can be very helpful for
people to implement it. A JSP sample is being constructed and will be available soon. In the future, we also plan to include CGI and ActiveX samples into the release package. Currently, the MLPQ server picked up 5 functions from the desktop system. There are still many other useful functions such as zooming, area calculating, buffering, and 2-D animation [Spec03]. To make those useful desktop functions also available in web applications is a work we plan to do in the future.
Bibliography


Appendix

APPENDIX A SOFTWARE PACKAGE LIST

mlpq_nt.exe : The M w3Socket component. LPQ web access system.
mlpq_nt.cfg : The MLPQ web access system configuration file.
client.exe : Client program in C/S system.

wwwroot1.zip : The HTML and ASP codes for the police emergency website. This package uses

W3Socket.zip : The w3Socket software package provides a method to access Socket in ASP. It is a shareware software by Dimac Development. It is also downloadable at “http://www.dimac.net/”.

wwwroot2.zip : The HTML and ASP codes for the police emergency website. This package uses ActivSocket component.

asocket.exe : The ActivSocket software package provides another method to access Socket in ASP. This is a 30-days trial version by Activxperts.com. It is also downloadable at “http://www.Activxperts.com/”.

police.txt : The constraint database for the police emergency application.

mlpq_code.zip: The source code of the MLPQ Web Access system.
APPENDIX B    MLPQ WEB ACCESS SYSTEM USER’S INSTRUCTION

1. Requirements

*Hardware:*

*Server Machine:* Windows 98, Windows NT 4.0, Windows 2000 Professional or higher, Windows XP Professional or higher.

*Client Machine:* No specific requirement.

*Software:*

*Server Machine:*

MLPQ/PReSTO system can run on Microsoft Windows 98, Windows NT4.0, Windows Me, Windows XP. To provide web service, any host running MLPQ Web Access System must support TCP/IP and activate IIS. The socket port of the MLPQ Web Access system is 2222. The sample police emergency application must run on Windows 2000 Professional with SP3, IIS 5.0 and PWS 5.0. To access socket in ASP asks for some special techniques. There are several software products can do this work. The sample application uses W3Socket for this purpose. It is a shareware product that comes from Dimac Development Company. Activxperts.com provide another choice. Their ActivSocket package is a very good product. It has enough documentation for the user. But it is not free.

IIS and PWS can be found from the Windows Installation CD.

A free w3Socket 1.1 can be downloaded from "http://www.dimac.net/"
A 30-day free trial version ActivSocket 2.1 can be downloaded from "http://www.activxperts.com/"

_Client Machine:_ Microsoft Internet Explorer 5.0 or higher.

2. _Server Installation and Implementation_

1) Get the MLPQ Web Access package and extract mlpq.nt.exe to any local directory.

2) Extract w3Socket.zip and install it to any local directory.

3) Install **IIS** (Internet Information Server) and **PWS** (Personal Web Server) if they are not available in your machine. IIS 5.0 is not a default installation in Windows 2000 Professional. But they are all included in your Windows Install CD. You can install them from “Start” → “Settings” → “Control Panel” → “Add/Remove Programs” → “Add/Remove Windows Components”. You can find the IIS option from the component list box. Selecting IIS from the box and click “Details”, you can find “Personal Web Manager” item in the component list. Make sure you already checked it and click “OK” to continue the installation. When installing PWS, please use the default settings and set the root directory to “c:\inetpub\wwwroot”.

4) If you already installed **IIS** and **PWS** but the root directory is not the same as we asked. You can change the root setting through the PWS manager, which can be loaded from “Start” → “Settings” → “Control Panel” → “Administrative Tools” → “Personal Web Server”. There are three choices on the left side of the PWS window. Please select “Advanced” from them. Or you can also select it
from the “View” menu. In the advanced view, you can see a “Virtual Directory” box on the top and three buttons on the right of the box. Select “<Home>” item in the box and click button “Edit” to change the root directory. You can also do all these settings through IIS management tools. You can find IIS at “Start” → “Settings” → “Control Panel” → “Administrative Tools” → “Internet Information Server”

5) Extract wwwroot.zip to “c:\inetpub\wwwroot”.

6) Create a sub-directory “\dbs” under “c:\inetpub\wwwroot” and copy police.txt to the directory.

7) Run mlpq_nt.exe and select “listen” item from its “help” menu.

8) Make sure the PWS service already started up in your server. To check the status of PWS, you can open the PWS manager from “Start” → “Settings” → “Control Panel” → “Administrative Tools” → “Personal Web Server”. If you are using Windows XP Professional, there may be no PWS icon under “Administrative Tools”. You can access it through IIS settings.

3. Client Execution

1) Open IE in client machine.

2) Input the Server's IP address followed with “login.html”. For example: “http://127.0.0.1/Login.html”.

3) Input a user name to login.

4) Do what you want by the function buttons in the bottom.
5) If you run the mlpq_nt.exe and the browser at the same machine, please make sure the MLPQ window are not covered by any other windows when you are querying. Otherwise, it may return blank graph. You can minimize the MLPQ window before you do the queries in your browser. The MLPQ will automatically popup after you sent out your queries and make sure you can get the correct answer.