TITLE:

Distributed Query System using JXTA

SOURCE:

Networks and Infrastructure Research Lab

AUTHORS:

Name	Organization
Kabe VanderBaan	Networks and Infrastructure Research Lab, Illinois
Kyle Bailey	MSU Capstone
Justin Crawford	
Mohamed Elias	
Tisha Melia	
Brett Schwarz	

DATE:

Feb 2, 2005

Version 1.5

Revision History

Version	**Date**	**Change Description**
1.0	1/19/2005	Initial release
1.1	1/19/2005	Added testing requirements and page numbers
1.2	1/25/2005	Updated Outline
1.3	1/31/2005	Edited Architecture & Design, Development tools, and Query Response
1.4	2/1/2005	Added drawings, screenshot, reorganized some segments.
1.5	2/2/2005	Added Sequence Diagrams, fixed alignment, etc.

1 Problem Description

In a network environment, a user desires to discover certain resources that exist on the network. These resources may be files, mobile devices, cameras, or any type of object or entity that exists in the network. The user might want to find the location of all digital cameras with a battery life of less than 10% so that he can request new batteries from the purchasing department. The user might also want to find a copy of a certain video file that is in an unknown location on the network. The user needs the ability to send out arbitrary queries across the network without the assistance of a centralized resource server. This can be accomplished with a peer-to-peer (p2p) network.

2 Solution

This solution defined in this project is an implementation of a distributed query tool that utilizes peer-to-peer technologies. This solution will be developed on top of JXTA, Java's p2p framework. The core service will be capable of receiving ANY query in a specified format. Each peer that receives the query will respond accordingly if the peer can understand the query, and the peer meets the requirements of the query.

The queries will be temporal in that they will stay valid for specified amount of time. All existing peers will respond to the query when the query is first created, and all peers that enter the p2p network will respond to the query if the query is still valid.

A client will be developed that utilizes the QueryService A Graphical User Interface (GUI) will be included on the client that will respond to user input, and pass a query to a generic QueryService that is capable of sending an arbitrary query and receiving the arbitrary results.

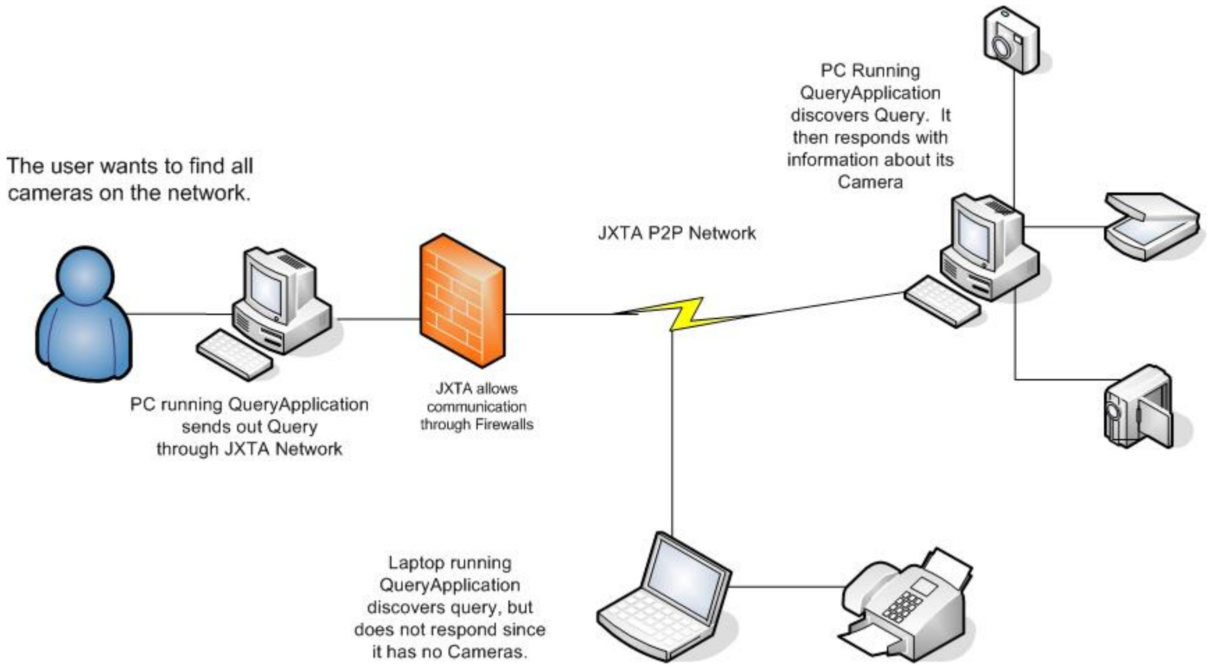

The user wants to find all cameras on the network.
PC running QueryApplication sends out Query through JXTA Network
JXTA allows communication through Firewalls
JXTA P2P Network
PC Running QueryApplication discovers Query. It then responds with information about its Camera
Laptop running QueryApplication discovers query, but does not respond since it has no Cameras.

Figure 1 - Solution Overview

3 Main Processes

The application is divided into three asynchronous tasks that rely upon communication between the tasks and other peers. These tasks are Query Generation, Query Reception, and Result Processing. Due to the instability and unreliability of individual connections, the application must be prepared to receive results at long time intervals, and cannot afford to wait for a result in some instances.

3.1 Query Generation

The Query Generation is done by the user interaction with the GUI. In the later, more complicated GUI, this may be by clicking on a room in a map, or in the case of the simple GUI, it may be actually typing out an XPATH query. The result of the GUI interaction is that the *read* method is called in the JXTAMetabaseService. The JXTAMetabaseService handles all JXTA network issues, so to the end user, the application resembles an actual database. The JXTAMetabaseService then sends the appropriate information to the QueryAdvertiser. This process could be thought of as belonging to the client in the traditional client/server model.

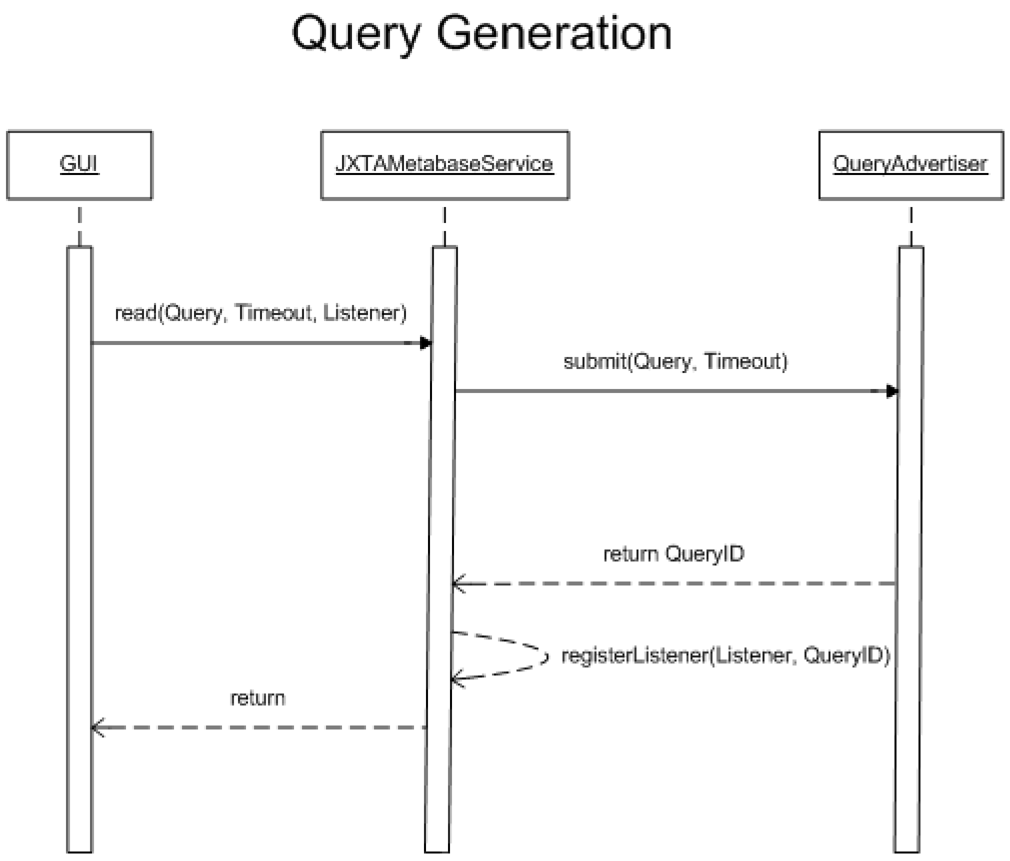


Figure 2 - The sequence of a QueryGeneration

3.2 Query Reception

Query Reception is handled by the QueryDiscoveryThread. It is constantly polling for new network queries including queries sent by its own GUI. If a valid query is found, it will be passed to its local JXTAMetabaseService, which will check to see if it has any resources that meet the query requirements. If there are any that do, they will be sent to the QueryResponder which will use JXTA Pipes to send the resource information back to the generator of the response, who may be on the same computer, or across the entire network. This process could be thought as belonging to the server in a traditional client/server model.

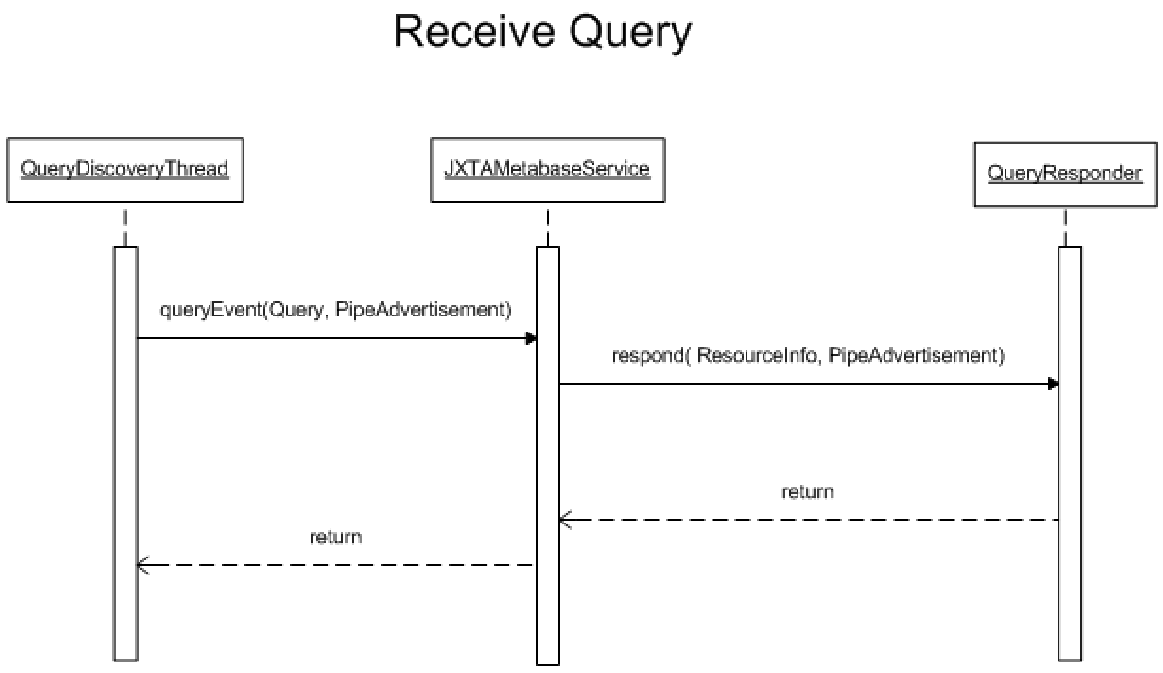


Figure 3 - A Sequence Diagram of Query Reception

3.3 Result Processing

Once the advertisement has been sent out to the network, the QueryResponseHandler must be ready to accept pipe connections from those that have resources that meet the query requirements. When messages are received through the InputPipe created by the QueryResponseHandler, the contents of the messages are extracted and sent to the JXTAMetabaseService, along with the QueryID of the message so that the results can be sent to the correct listeners. When the JXTAMetabaseService receives the results, it then sends the resulting information (the goal of the original query) to all the interested GUIs, so they can display the information in whatever format they prefer. This process would belong to the client in the client/server model.

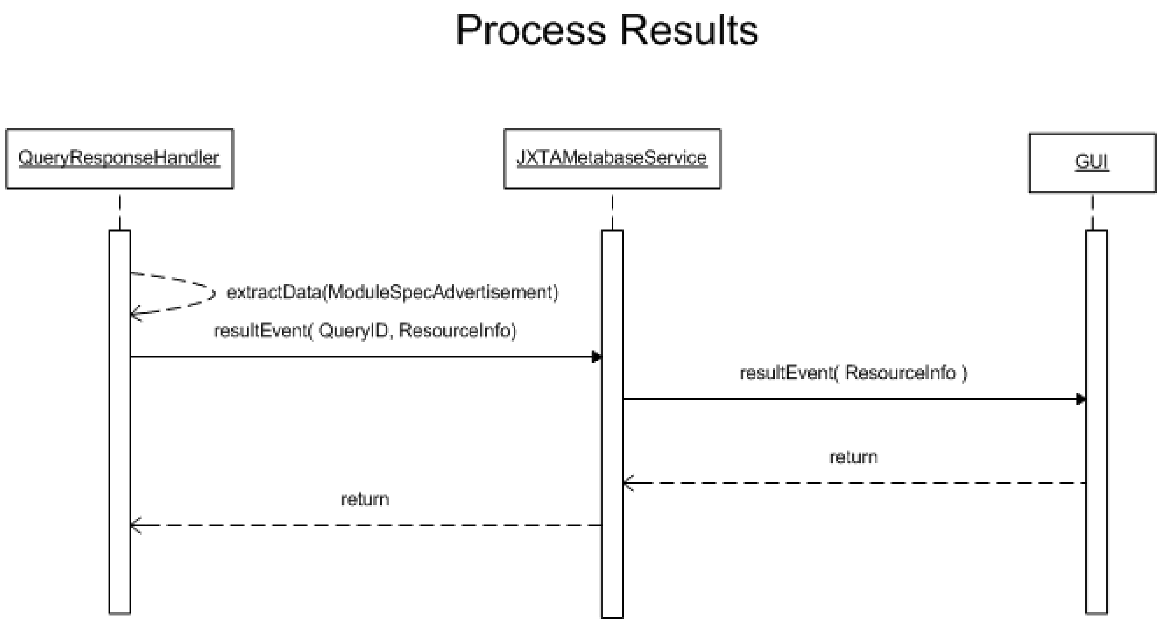

Process Results
QueryResponseHandler
JXTAMetabaseService
GUI
extractData(ModuleSpecAdvertisement)
resultEvent(QueryID, ResourceInfo)
resultEvent(ResourceInfo)
return
return

Figure 4 - A Sequence Diagram of Result Processing

4 Architecture and Design

The Design Architecture consists of one main application that can be viewed to have two separate parts. The Query Generation (client) side is used to request information about resources on the network, while the Query Response (server) side is used to respond to these arbitrary queries. The application is present on all systems in the network that are involved with the p2p network. For this application, this will run only on PCs with the Java 2 Platform.

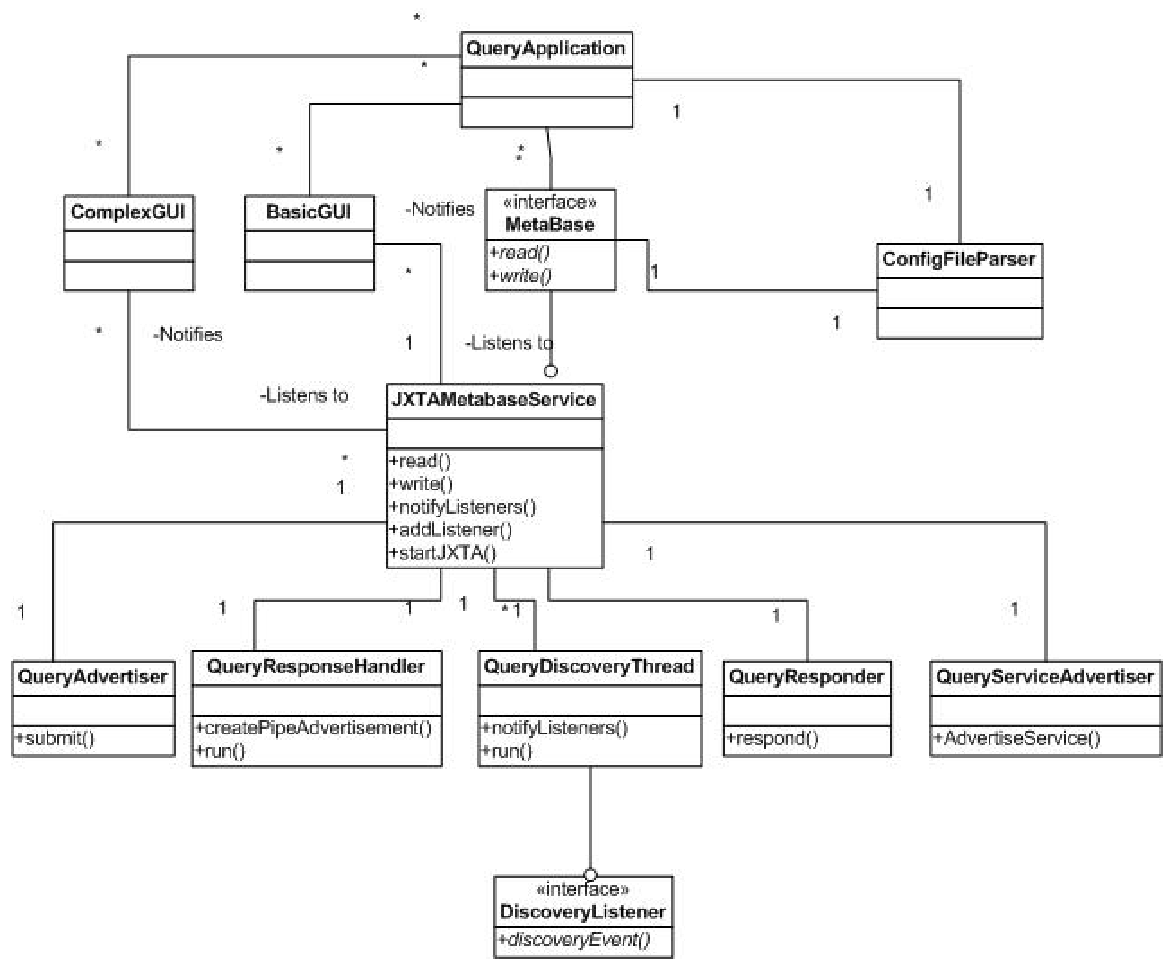


Figure 5 - Class Diagram for DQS Solution

4.1 *QueryApplication*

The main class for this project is the QueryApplication. It will load the JXTAMetabaseService and the GUI depending on the configurations set in the configuration file. The QueryApplication will interact with the metabase through the standard Metabase interface with a few possible additions. The

QueryApplication will contain a ConfigurationFileParser to parse the configuration file specified by the -c command line argument.

4.1.1 Graphical User Interface (GUI)

The user will interact with the application through a graphical user interface built on the Java Swing platform using Model View Controller (MVC) principles. There will be two versions of the GUI throughout the development of this project.

4.1.1.1 Basic GUI

A basic GUI will be used; it will serve the purpose for simulation or debugging only. The basic GUI will contain a textbox for the specified query, a textbox for the query duration; a submit button to send the query out to the JXTAMetabase, and a list of results (listbox) that will show results from the network search.

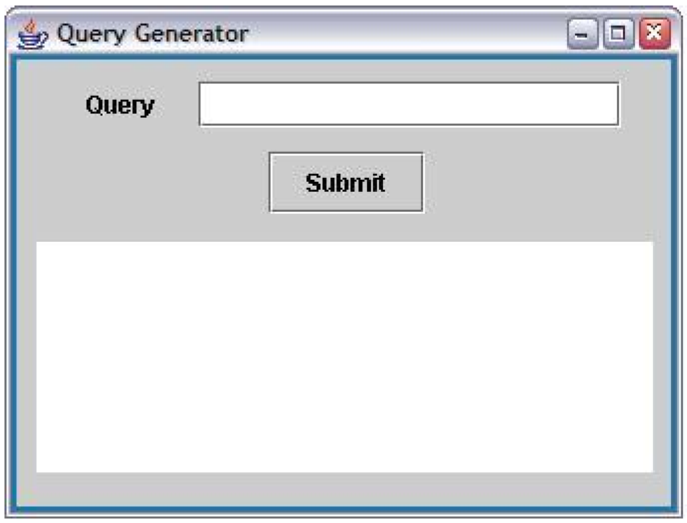
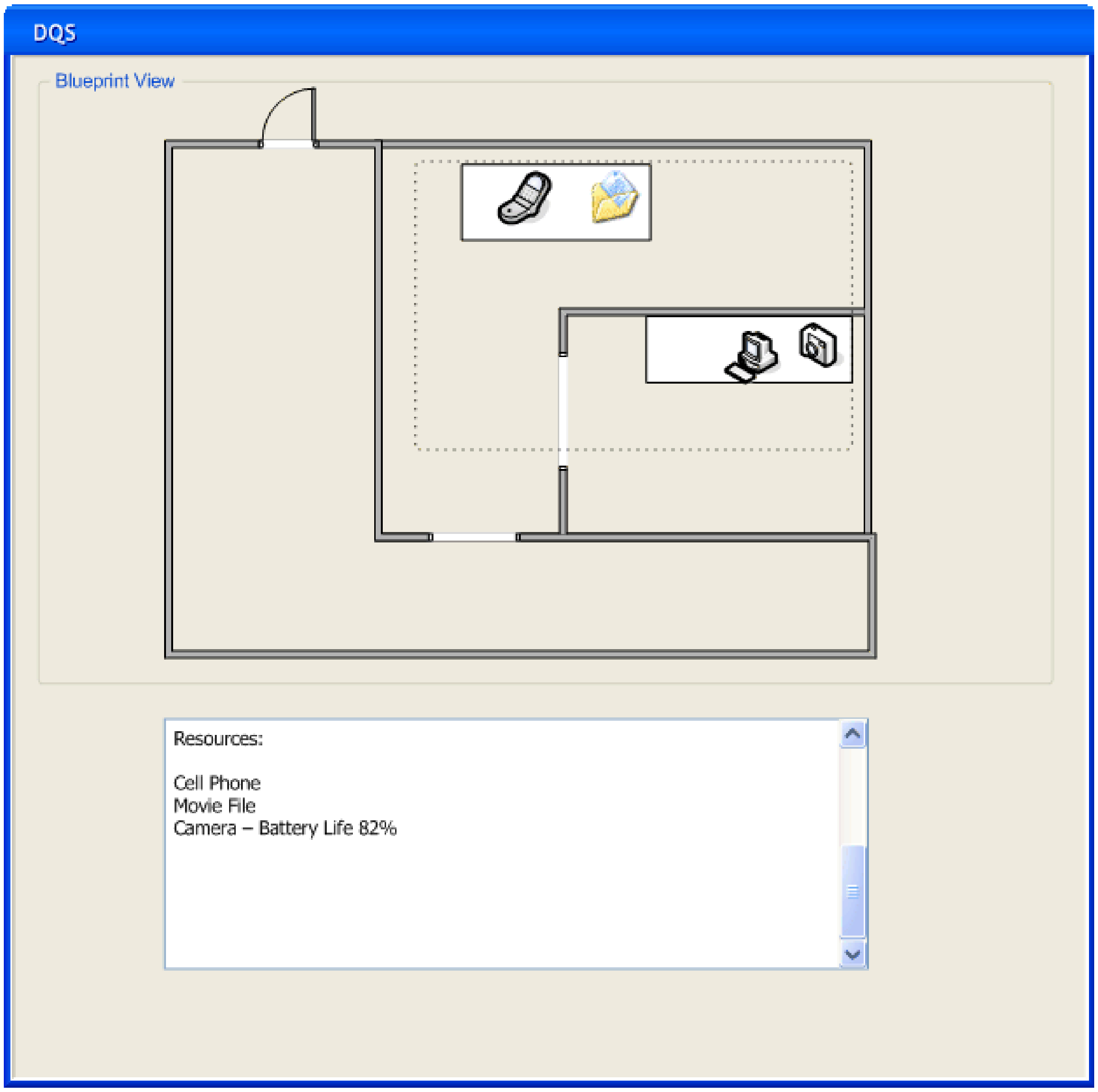


Figure 6 - An Example Basic GUI

4.1.1.2 Final GUI

This final version will be more sophisticated than the basic version. It will read in a script and output a blueprint of a room/building and takes user input using mouse-clicks. All the peers located will be shown on the blueprint in their exact location in the grid, allowing users to view movement of resources as well. Blueprint information will be located in the configuration file.



Resources:

Cell Phone
Movie File
Camera – Battery Life 82%

Figure 7 - An Example Complex GUI

4.1.2 JXTAMetabaseService

Much like a database, a metabase is used to store and retrieve information about various resources. The JXTAMetabaseService is used to add on additional functionality to allow database-like access to an online community of resources.

Interaction with the metabase is done with the standard metabase interface that is used by other Motorola applications. The interface includes multiple *read* and *write* functions, but new additions will be necessary due to the nature of the p2p network.

The JXTAMetabaseService will start up a QueryDiscoveryThread and will be set up to listen for events generated by this thread. When an event occurs, the JXTAMetabaseService will use the XPATH string from the query to search its own collection of resources. If any resources meet the query requirements, it will then start the process to send the information back to the clients that requested the resources.

The JXTAMetabaseService will also notify GUI listeners when it receives responses to the queries it generates, so that they can display the information in whatever format they choose. It will be able to keep track of multiple queries from multiple GUIs and give the corresponding results to the GUI that generated the query. This will be done through unique QueryIDs that are generated with JXTA functions.

The JXTAMetabaseService will have certain configurable features that can be modified through the configuration file. These features include:

-Frequency for the DiscoveryThread to look for new Queries
-Standard Timeout values, etc

4.1.3 ConfigurationFileParser

The QueryApplication contains a class that will load in a configuration file. This file is explained in detail in the next section. The ConfigurationFileParser reads through the XML configuration file, adds resources to the metabase, configures the metabase and configures the user interface. Metabase configuration information will be passed to the Metabase through the standard write method, along with resource information. GUI configurations will be sent back to the QueryApplication so it can load the corresponding GUI.

4.1.3.1 Configuration File

The Configuration File will be an XML document that contains information about resources and configuration information for the GUI and MetaBase. It must contain a <config> tag to be a valid configuration file recognized by the application. The specified format is as follows:

```
<config>
        <metabase>
        ...
        </metabase>
        <gui>
        ...
        </gui>
        <resource>
        ...
        </resouce>
        <resource>
        ...
        </resource>
        ...
</config>
```

There can be 0 or 1 metabase configuration tags, 0 or 1 gui configuration tag, and 0 or more resource tags, but there MUST be a config tag to be valid.

An example resource definition is as follows:

```
<RESOURCE>
  <CAMERA NAME="199.104.137.91/18000" MAKE="Canon" MODEL="VC-CR4" />
</RESOURCE>
```

4.1.3.2 Command Line Arguments

The program expects to receive "-c filename" from the command line to specify the Configuration File location. If the user did not enter –c or did not provide a valid file, the program will output an error message and exit.

4.2 Query Generation

4.2.1 QueryAdvertiser

The QueryAdertiser is responsible for sending queries out to the world. It creates a JXTA Module Specification Advertisement that contains the desired XPATH query, and a Pipe Advertisement so that the client that receives the notification can communicate with the sender. Each Query gets a unique ID that it sent in the Advertisement. The Query String itself is located in the Description portion of the Advertisement.

4.2.2 QueryServiceAdvertiser

The QueryServiceAdvertiser is a feature that would be used to advertise the QueryService itself using the JXTA Service implementation. It is currently not going to be used, but may end up being developed. Using this method, anyone on the network that knows about this service could receive an interface to use the service without having it locally on their machine.

4.2.3 QueryResponseHandler

The QueryResponseHandler runs in the background as a thread. It listens to the input pipe created upon initialization for any responses to the queries that have been advertised. A response takes the form of a message sent through a JXTA Pipe. Pipes can be thought of as unreliable sockets, as they do not guarantee delivery. If the message is valid, the meaningful data is extracted and forwarded to any listening JXTAMetabaseServices.

4.3 QueryResponse

QueryResponse consists of a group of classes and files that is used by a device to respond to the incoming query.

4.3.1 QueryDiscoveryThread

The QueryDiscoveryThread is a thread that is started by the JXTAMetabaseService. Its purpose is to listen for any queries generated in the JXTA network, such as a request for all cameras within a certain distance from a location, and to notify the JXTAMetabaseService of a valid query. These queries are JXTA Service Advertisements that contain a formatted XPATH query and information to establish a JXTA pipe connection to the originator of the query. If a valid query is found, it notifies all listening metabases with all information needed to handle the request.

4.3.2 QueryResponder

The QueryResponder uses the JXTA pipe mechanism to transfer information from the local metabase to the client that generated the query. The QueryResponder establishes a pipe to the originator with the pipe information from the JXTA Module Specification Advertisement that was received by the QueryDiscoveryThread. Once the pipe is opened, a message containing the description of all resources that meet the query requirements is created, and sent across the pipe.

5 Simulation

Simulations will occur within the defined project to show the power and flexibility of the system. The simulation will show a number of devices that are moving around a building. It will simulate one or more scenarios that will be outlined.

5.1 Actors

An actor is a mobile device that is available in the network and will be used to simulate the identified scenarios. Actors will be defined in the Configuration File in XML format.

5.1.1 Mobile Strategies

Various strategies will be implemented to show the movement of actors. For instance, a “stand still” strategy will be a strategy where the device never moves. Another strategy might be a 15 second strategy, where every 15 seconds an actor moves. Mobile Strategies will be defined in the Configuration File in XML Format.

6 Development Tools

This section describes the development tools utilized in the development process.

6.1 *Java 2 SDK*

The Application will be written entirely in Java 2, using the latest SDK, version 1.4.2_06.

6.2 *Concurrent Version System (CVS)*

CVS is used to keep track of all work and changes in all files. It allows several developers to collaborate in implementing the same project and share their files.

6.3 *Sandbox*

Individual sandboxes are set up in every computer used in the development process. These sandboxes will keep the local copy of files in each workstation. The developer can update, upload, or download files between their local sandbox and the CVS server.

6.4 *Integrated Development Environment (Eclipse)*

The Eclipse IDE is an open source development environment suited for Java Development.

6.4.1 XML Buddy

XML buddy is used as a plug-in to Eclipse to view and edit xml files.

6.4.2 Visual Editor Project

The Visual Editor Project is used as a plug-in to eclipse to create the graphical user interfaces. It allows WYSIWYG editing with Swing, AWT, and other Java GUI Packages.

6.5 *Ant*

Ant is to be used for all compilation and scripting. The ant file that is created must be able to integrate into the Eclipse IDE. This allows more portability for building the application.

6.6 *Java Archive (JAR)*

A Jar file is much like a zip file. It is a mechanism to store all java class files in a single file for distribution. Applications can be execute using a Jar file. Mechanisms to generate JAR files must exist in the ANT scripts. This will allow easy distribution of the application once finished.

6.7 JUnit

JUnit will be used to assist in testing. JUnit is an open-source testing tool that allows for the development of scripted test cases in Java.

7 Requirements

This section descries the expected functionality of the Distributed Search application. The minimum requirements need to be met for this to be a successful project. Milestones are key deliverables that will be met along the way.

7.1 Minimum Requirements

Minimum requirements will be filled out shortly.

7.2 Milestones

Milestones are the deliverables that will be completed as the project continues forward. The identified milestones are a guideline to follow and may change as the project to follow. A milestone will be considered complete when ALL the criteria for the milestone are complete. It is impossible to have a later milestone completed before a previous milestone (i.e., milestone 2 must be completed before milestone 4 is completed). The Milestones will be filled out shortly.

7.2.1 Milestone 1

1) Rename JXTAQueryService to be JXTAMetabaseService
2) Implement the MetaBase Interface in JXTAMetabaseService
3) Create the following classes: (I might be missing a few)

QueryDiscoveryThread
QueryResponder
QueryAdvertiser
QueryServiceAdvertiser
QueryResponseHandler
ConfigFileParser

4) The QueryApplication should take in a "–c filename" command line argument if the argument is not there it should exit
5) Discover local peers utilizing advertisements.
    a. Join the netpeer group
    b. Get the Discovery Service
    c. Get the Pipe Service
    d. JXTAMetabaseService should implement the DiscoveryListener Interface
6) Build and execute the project using Ant
    a. Compile the edu.msu.dqs.QueryApplication class by setting the correct src parameter
    b. Compile to the bin directory
    c. Execute from the bin directory
    d. It should execute QueryApplication
7) To show that this works,
    a. Execute 5 instances of this application, on at least 2 machines

   b. We should see discovery messages from all 5 peers

8) Outline of the technical specification.
9) Pass Strings and output Strings indicating that the classes are bing called in the right order (System.out.printlln)
10) 3 scenarios to show
    a. A Query
    b. Adding Resources
    c. Responding to Query

7.2.2 Milestone 2

GUI

1) Has Textbox to input String
2) Has Textbox to input Query Timeout
3) Has a List to output responses
4) Has Submit button
5) Has Clear button
6) Has Cancel button
7) Registers listener with Metabase
8) Submit button sends query to JXTAMetabaseService
9) Clear button empties the list.
10) Cancel button does nothing at this time.
11) Populates responding queries in list.
12) Exits cleanly

JXTA Metabase Service

1) Query Advertiser is implemented as a JXTA Service.
2) Query Advertisements have timeout.
3) Query Advertisements propagate to others in the peer group.
4) Query Discovery Thread polls for new queries.
5) Peers compare Query Advertisement against each document.
6) Query Discovery notifies Query Responder when Document matches query.
7) Query Responder sends matching documents to Querying peer over Pipe.
8) Notifies registered listeners whenever it receives a response for a query.
9) Unique QueryID is generated per Query
10) Queries are cached by QueryIDs

Query Advertisement

1) Payload is XPATH string and queryID

Config File

1) In new format
2) Query Discovery Thread refresh rate is configurable
3) GUI is configurable
4) Metabase is configurable

Documentation

1) Outline for Architecture document is complete

Ant

1) Ant is capable of executing 5 nodes for test case
2) Ant can perform clean function

7.2.3 Milestone 3

To be described at a later date.

7.2.4 Milestone 4

To be described at a later date.

8 Testing

JUnit will be used to help with testing purposes. All the test cases will be included in JUnit.

All test cases will be documented and stored to ensure that the Application meets all requirements with every release.

QueryApplication Test Cases:

1) -c argument does not exist
2) –c filename does not exist

More to come as application progresses