1 The Metadata Engine - Developed

1.1 Introduction

The purpose of this chapter is to outline in detail the prototype that was developed as proof of concept that a metadata engine is capable of distributed processing over a network. The previous chapter outlined the five different models that could have been used to develop a metadata engine along with two methods to store the access metadata. This chapter will describe in much more detail the specific data model and access metadata storage method that was used in the prototype. The chapter also describes in detail the Isite software package that was used as a base for the prototype along with the modifications that were made to its source code in order to get it to function as a metadata engine.

1.2 The Adopted Data Model

Of the five data models that were outlined in the previous chapter, option 5 was the model that was chosen to develop the prototype. The main reasons why this approach was taken as compared to the other four approaches were:

1) In option 1 the ANZLIC compliant and access metadata are stored on the users local server along with the metadata engine. This was deemed to be unsuitable since it is a waste of resources to have copies of the metadata duplicated on every server that wishes to access the distributed databases. Not only is this a waste of storage space, but it is also a nightmare to keep up to date. Every time that a metadata record is changed, added or deleted by a data custodian this alteration has to be made to all copies of the metadata record located all over the network.

2) In option 2 the ANZLIC compliant and access metadata are stored on the data custodians server and the network addresses of each of these data custodians servers is located on the users local server with the metadata engine. This is a better option than the first as there is minimal duplication of

metadata across the whole of the network. Essentially only the data custodians have a copy of the metadata on their servers. There is one main problem associated with using this approach. The problem (the same problem also exists with option 1) is that there will be a copy of the metadata engine on every server that uses the system. Not only does this make it difficult to upgrade all the individual engines with information concerning new servers that have recently been added to the system, it also seems wasteful of resources when a single metadata engine, that has the ability to serve all users needs, could be located at a central location. A single central metadata engine would be considerably easier to maintain.

3) In option 3 both the ANZLIC compliant and access metadata are stored on a single central server along with the metadata engine. This option doesn't have the problem of the option 2, as there is only one copy of the metadata engine in existence on a central server. It also doesn't have the problem of option 1 where there is a large duplication of metadata, although there is some duplication of metadata as both the central server and the metadata custodians themselves have copies. The problem that is associated with this approach is that the data custodians lose control of their metadata. Once the metadata has been entered into the system at the central server it is no longer directly controlled by its original custodian. Not only will many custodians have a problem with this, due to the "information is power" attitude that many have, but it will also make it difficult for the custodians to actually keep the metadata records up to date. For this reason it was decided that an approach where the data custodians maintain control over their own metadata would be best.

4) In option 4 the metadata engine is located on a central server along with the addresses of all the remote servers that contain the individual custodians metadata. In this case the individual data custodians have direct control over the metadata that is searchable by the metadata engine located at the central server. This option has none of the disadvantages of the first 3 options, as there is no duplication of metadata, no update problems for the metadata, and no custodianship problems. There is however one small problem with this

approach. The prototype metadata engine that was developed was intended to act not just as a distributed spatial data processing system, but also as a data directory. The system was intended to supply metadata for all the spatial datasets that are in existence. It is extremely likely that in the early stages of development of a commercial system, many data custodians will not have the means to serve up not only their spatial data, but also their metadata. To solve this problem option five was developed. It had all the properties of option four, except it allowed for metadata owned by a custodian who did not have the ability to serve it on the system, to have a copy of it located on another server (including the central server). This gave the system the ability to have all metadata for all datasets within it, even if all the custodians do not have the ability to serve the data and metadata themselves. In the instance where there is a metadata record for a dataset that is not accessible for distributed processing, the access metadata will contain metadata to this effect and the metadata engine will inform the potential user of this fact. The metadata record can then be viewed by the potential user to find out how to get access to the data. Therefore option five was adopted.

1.3 Adopted access metadata data storage method

Of the two options outlined in the previous chapter detailing how to handle the access metadata, the file type was adopted for the developed prototype. There were several reasons for deciding upon this approach, including:

1) The fact that there is no one standard format for metadata that appears in pages lower than page 0. This fact alone makes the coding of a prototype difficult as the resultant prototype will have to search through possibly several metadata page levels looking for certain pieces of metadata to return to the user.

2) Contrast the aforementioned difficulties above with the relative ease of implementing a separate file approach. In this approach the actual metadata that gives the user information on how to access the dataset is not only stored in a single file but also in a standard format within that file. Not only does this mean that coding the metadata engine that will extract the required metadata is much

easier, it also means the metadata is much quicker to find and extract due to the simplicity of the system.

3) A point that should be made with reference to developing a metadata engine using Isite (described in detail in the next section) is when Isite actually searches the metadata records that are located on a server it does not actually physically search the metadata records. Isite actually searches through an indexed list of words that appear in each of the records. Thus if you wished to undertake the pages approach you would actually have to develop a search engine that searches through each of the metadata records anyway. A search engine that searches through different metadata records of varying format is much more difficult to develop than a search engine that simply searches through a single file that contains all the access metadata, for all datasets at that location in a standard format.

4) A metadata engine developed using the pages approach would be the simplest to maintain once it has been developed. The problem is that developing such a system would take much more than a research project by one individual in such a limited period. The separate file approach is adequate to provide proof of concept. Figure 1-1 shows the architecture of the prototype.

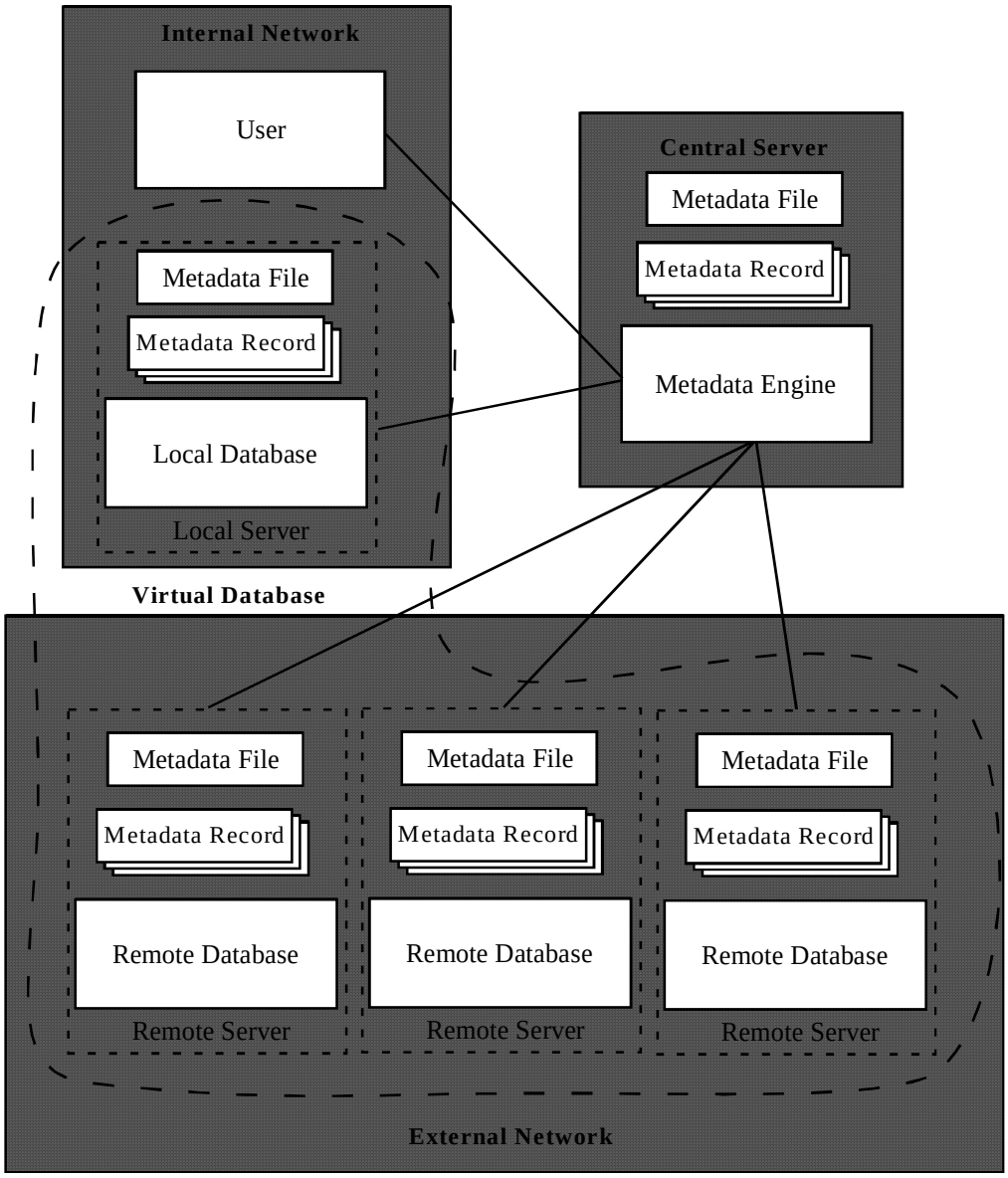

Figure 1-1: Architecture of the prototype metadata engine.

1.4 Isite

To develop the engine a freeware product called Isite was used as a base. Isite, in its unaltered form, allowed for a metadata search engine to be developed. It allows a user to search and view metadata records that are located across a network. Isite does not have the ability to display or query the actual datasets themselves.

Isite is a software package that uses the Z39.50 ANSI/NISO, a client/server based protocol for information retrieval (ANSI 1992), standard communication tools to

access databases that are stored on a distributed network. The software is public domain software, produced by the Center for Networked Information Discovery and Retrieval (CNIDR), and is freely available for anyone to download and set up on their site (Gamiel and Warnock 1994). The clearinghouse that has been established in the USA by the FGDC uses the Isite software as does the Australian Spatial Data Directory recently established in Australia by ERIN (FGDC 1996a, Hatton 1997a). The architecture of the Isite system is shown in Figure 1-2.

In Figure 1-2 it can be seen that there are two main methods for searching databases that are located on a remote server from your local system. The first method is simply to spawn a Z39.50 client from a local Unix machine which has Isite installed. This client will make contact with the remote server that is currently running a zserver session and then search the relevant database through a Search API (SAPI).

The second, and more convenient, method is to search the remote databases through a WWW browser interface. In this method an initialisation HTML form is used to establish a Http to Z39.50 stateful gateway with the remote server, running a zserver session. This once again allows the databases to be searched through the SAPI. "Stateful" means that a Z39.50 session is initialised only once and is interactively used by a stateless WWW browser. There is an inactivity timer on the gateway that automatically closes the session after a pre-defined amount of time of inactivity (Gamiel and Warnock 1994). Figure 1-3 shows the architecture of the stateful gateway.

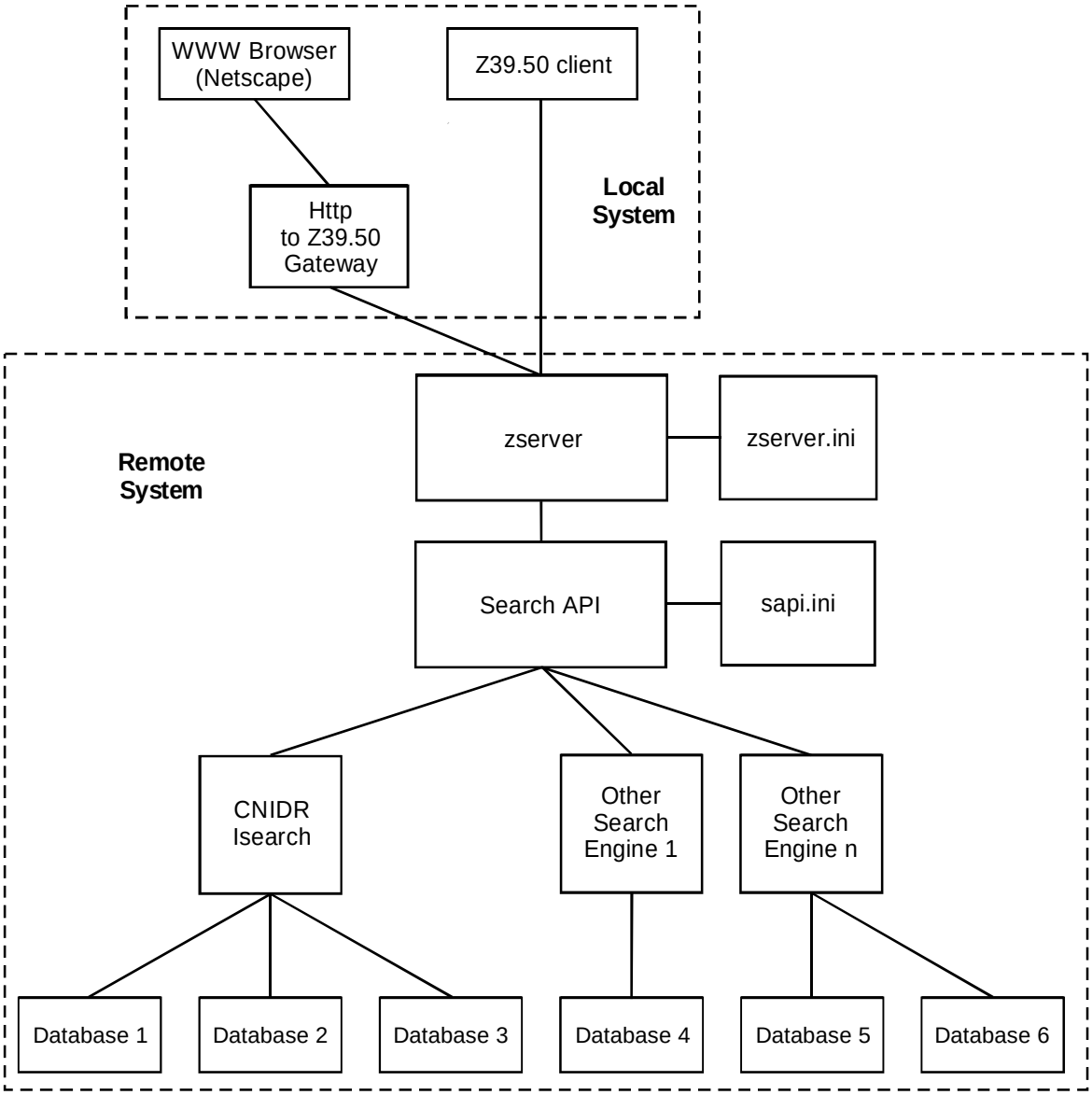

Figure 1-2: Isite Information System Architecture (Gamiel and Warnock 1994).

In Figure 1-3 the combination of the http server, zgate and zcon applications represents the gateway running on a single machine. The process that occurs in the event of the user of the WWW browser submitting a query is as follows:

- The WWW browser connects to the http server and posts a HTML form containing information related to a new or existing Z39.50 session.
- The zgate CGI application then parses that form and either starts a new zcon process or connects to an existing process if one already exists.

- The user's request is then passed from zgate to the appropriate zcon, which in turn communicates with the remote Z39.50 server.

- The remote Z39.50 server passes the results from the query back firstly to the zcon process, and then in turn back to zgate, the http server and finally the WWW browser where the original query was made.

- At the WWW browser the results are then displayed for the user to view.

- The zgate CGI process then exits, however the associated zcon process stays alive, holding the Z39.50 connection open until a predefined period of inactivity is exceeded, upon which time the zcon process exits (Gamiel and Warnock 1994).

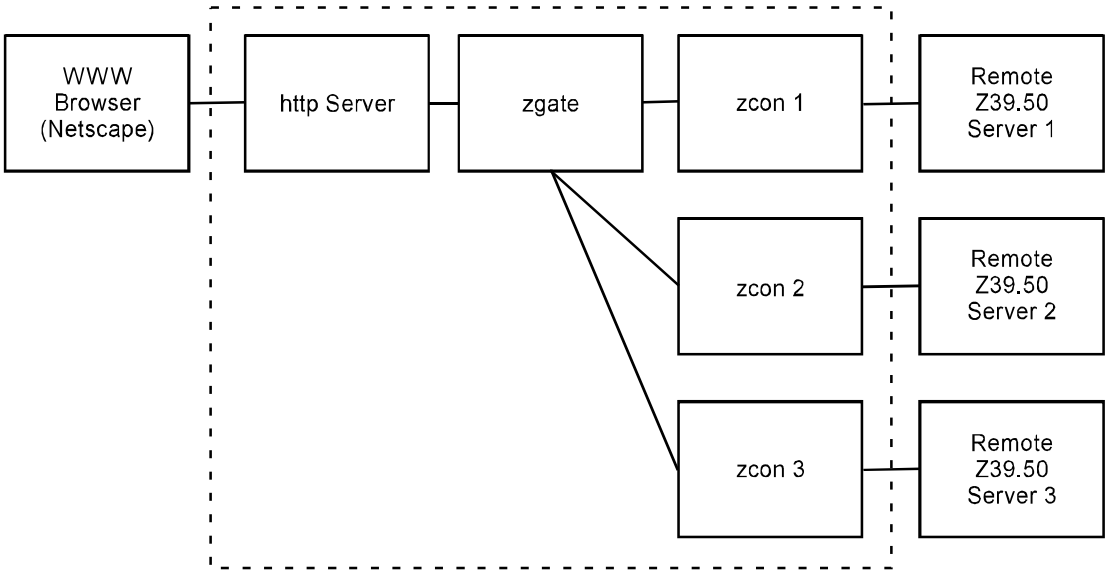

Figure 1-3: Http to z39.50 stateful gateway architecture (Gamiel and Warnock 1994).

The applications that are possible when using Isite require accessing text search facilities. For this reason CNIDR developed a SAPI which generalises access to arbitrary database systems via a common API (Gamiel and Warnock 1994). Any application that links with this API inherits the functionality of any database system that may reside "behind" the API. For the system administrator this requires the maintenance of one or more text files that describe the databases that are currently available, their location on the file system, etc (Gamiel and Warnock 1994). The users

of the system they should be unaware of the details behind the search. The aforementioned databases are in fact databases that contain the metadata records for all the datasets that exist for an organisation.

1.5 The access metadata file

The file that contains the metadata that instructs the remote user on how to gain access to the datasets available for that server is called "directory.txt". The file is located in the same directory as the zserver executable and entries for each dataset have a format that is as shown in Figure 1-4. Figure 1-4 is an example of what an entry for a planning dataset may look like in the directory.txt file.

A description of the function of each of the lines in Figure 1-4 is follows:

DataRecord - This is a marker that signifies that a new metadata entry is about to begin.

HTMLRecord - Consists of two lines. The first line is the marker, **HTMLRecord**, and the second line is the actual name of the ANZLIC compliant metadata record that corresponds to this entry.

DataDescription - Consists of two lines. The first line is the marker, **DataDescription**, and the second line is a description of the dataset, that this entry corresponds to.

DataDescriptor - Consists of two lines. The first line is the marker, **DataDescriptor**, and the second line is the name of a variable that is used to keep track of the input stream from that dataset.

DataRecord

HTMLRecord

Planning.html

DataDescription

Planning Data

DataDescriptor

planning

DataLocation

surprise.sli.unimelb.edu.au

DataBaseCallString

/opt/local/java/bin/java -classpath /opt/local/java/classes:/opt/local/java/lib/ classes.jar:/opt/local/java/lib/rt.jar:/opt/local/java/lib/i18n.jar:/opt/local/j ava/lib/classes.zip:/opt/local/java/lib/postgresql.jar:/usr7/www/cgi-bin/ Query_ JDBC jdbc:postgresql://surprise.sli.unimelb.edu.au:5432/planning

DataBaseTempFileName

/tmp/JDBCPlanTemp

LocatorName

Planning Unique Feature Identifier (surprise)

LocatorType1

Textfield

LocatorType2

ufino

Figure 1-4: Example of planning access metadata that appears in the directory.txt file.

DataLocation - Consists of two lines. The first line is the marker, **DataLocation**, and the second line is the actual location of the dataset.

DataBaseCallString - Consists of two lines. The first line is the marker, **DataBaseCallString**, and the second line is the actual call string that is required to be invoked to access the dataset.

DataBaseTempFileName - Consists of two lines. The first line is the marker, **DataBaseTempFileName**, and the second line is the name of a temporary file that stores the results of the query as they return from the remote server.

LocatorName - Consists of two lines. The first line is the marker, **LocatorName**, and the second line is the name of the variable that will hold the identification number that the data is to be centered around.

LocatorType1 - Consists of two lines. The first line is the marker, **LocatorType1**, and the second line is type of the variable **LocatorName**.

LocatorType2 - Consists of two lines. The first line is the marker, **LocatorType2**, and the second line is the name of the column in the table in the database that the search is to be conducted on.

It should be noted that each of the datasets that have ANZLIC compliant metadata records in the Isite metadata database will also have an entry in this file. If any of the datasets are not accessible they should have an entry in the file that gives the remote system this information. At this stage the prototype that has been developed does not cater for this occurrence.

1.6 How the Prototype works

The prototype that has been developed as part of this research allows the user of the metadata engine to search for relevant datasets based on keyword, spatial and temporal parameters using a simple web browser interface. The engine returns hyperlinks to the ANZLIC compliant metadata records that satisfy the users search parameters. The prototype not only allows for the ANZLIC compliant metadata records, for the each of the datasets that satisfied the search parameters, to be viewed, it also allows a user to view one or more of the actual datasets concurrently.

The datasets are viewed using an easy to use spatial data viewer, created by (Polley 1998) as part of his research into the two way dissemination of spatial data across the Internet. Polley's research is of particular interest to this thesis as it enables all five data types to exist on the system. At the present time type 1 (imported), type 2 (local public access) and type 3 (local private) data are the only types available. The system has no facility for modifying and updating the data that is being viewed. This means that data types 4 and 5 are not available. Polley's work investigates, among other things, the concepts that are involved in not only viewing data stored on another server, but also providing for the modification and update of those datasets. Once the two research projects are combined all five data types could be represented. A by-product of Polley's research was the development of a spatial data viewer that is able to draw points and lines. This was used to view the datasets.

The steps that are undertaken by both the user of the metadata engine and the metadata engine itself during a successful search are outlined below:

1) The user establishes an http to z39.50 stateful gateway between their machine and the server where the metadata engine is located. This is achieved by loading a web page that does the initialisation of the gateway automatically when the connect button located on the page is clicked. This page is located at the web address http://www.sli.unimelb.edu.au/~honey/gateway.html. The appearance of the web page is shown in Figure 1-5.

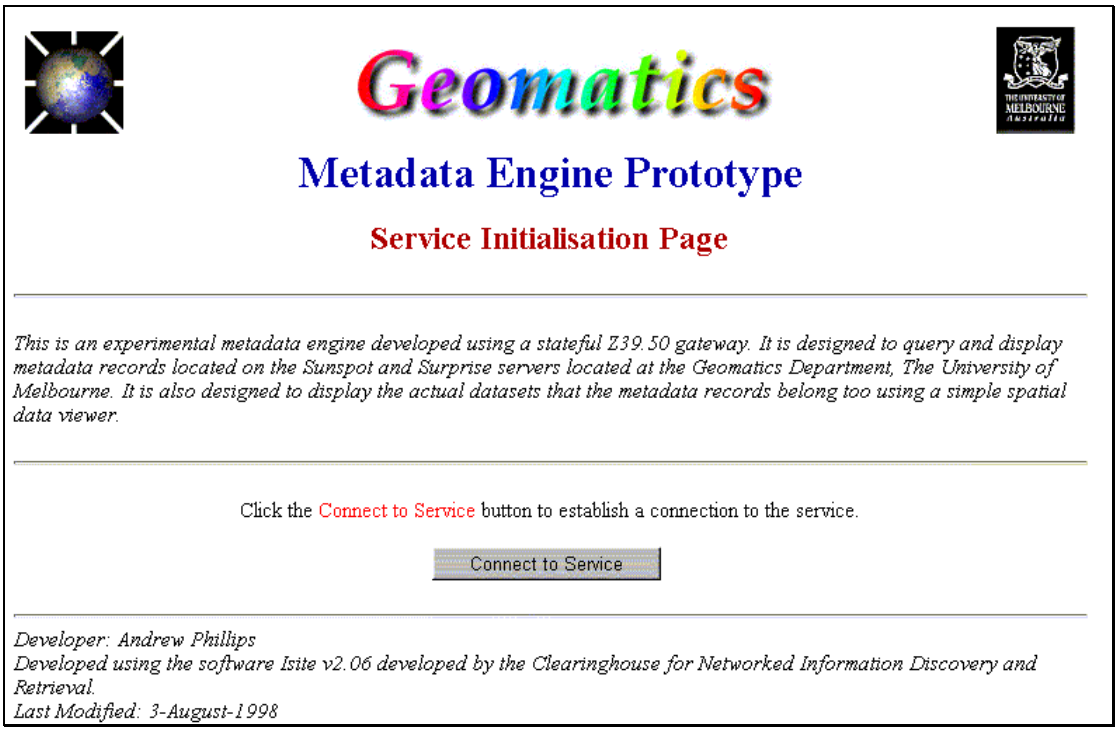

Figure 1-5: Screen dump of the gateway.html web page.

2) Once a stateful gateway has been initialised a search page is displayed on the users web browser. The search page allows the user to search for spatial datasets using keyword, spatial and temporal search parameters. Once the user enters the search parameters for the types of datasets that they wish to find they then click the submit button on the page to submit their query.

3) Once the query has been submitted the zserver application sends the search parameters to each of the remote servers that contain metadata databases. Whilst this is occurring it searches its own metadata database using Isearch and the search parameters supplied by the user to see if any local metadata satisfies the query.

4) On each of the remote servers Isite uses Isearch to search the metadata databases for metadata records that satisfy the search parameters specified by the user. When a metadata record is found that satisfies the search criteria two things occur:

   a) The access information for that dataset is retrieved from the access metadata file. Isite has been modified so as though it uses the metadata records name to search for the access metadata in the access metadata file. In the access

metadata file each of the metadata entries has the name of the ANZLIC compliant metadata record that it corresponds too at the top of the entry. The access metadata is simply retrieved by searching for the ANZLIC compliant metadata records name and then extracting all the following access metadata.

b) Once the access metadata has been retrieved it is encoded onto a return string along with a hyperlink to the ANZLIC compliant metadata record. A separate return string is encoded for each ANZLIC compliant metadata record that satisfies the search parameters. The return strings are then passed back to the metadata engine.

5) Upon receiving of the return strings from the remote servers the metadata engine decodes them. Whereas the encoding process merged all of the individual elements of the access metadata and the hyperlink into one string, the decoding process breaks the return string down into the individual access metadata components and ANZLIC metadata record hyperlink. Once the decoding of the return sting is complete the following two events occur:

   a) The access metadata is written into a file called "Directory_Andrew.txt" in a standard format. This file will be referred to by the data viewer to determine which datasets are to be displayed and how to display them.

   b) The hyperlinks to the individual metadata records are displayed on a web page. Each of the individual ANZLIC compliant metadata records can be viewed by simply clicking on the hyperlink with the mouse. Along with hyperlinks the web page also has a button that, when pressed, starts a simple spatial data viewer that enables the user to view all the datasets that satisfied the search parameters defined in their query.

6) The spatial data viewer when first started by depressing the "Bring Up Spatial Data Browser" button, has the appearance of Figure 1-6. The user is able to display the spatial datasets they wish by depressing the "Data Manager" button. By depressing this button a form/menu is displayed that allows the user to choose the datasets that they wish to view using the spatial data viewer.

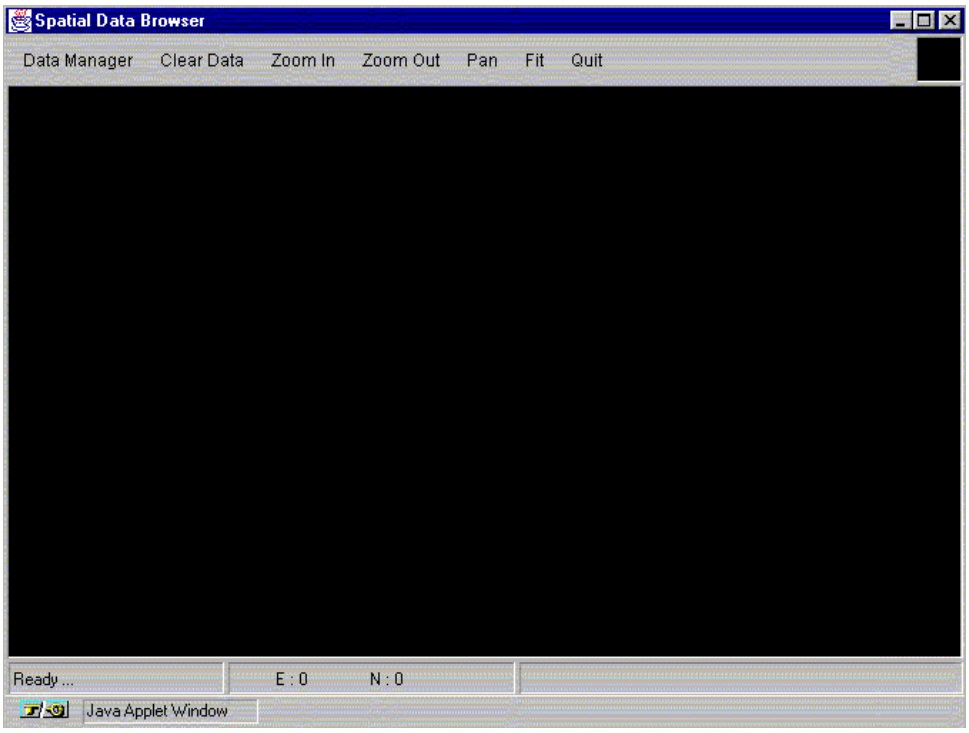

Figure 1-6: The spatial data viewer.

The form/menu was created by consulting the "Directory_Andrew.txt" file which was created when the return strings from the remote servers were decoded. The code that actually creates the menu scans this file to find the datasets that were returned as a result of the user's query. For each dataset that is returned a position on the menu is created for it. An example of the menu appears in Figure 1-7.

Check boxes along side the names of the individual dataset indicate which of the datasets are selected. A tick in the check box indicates that the dataset that it corresponds to has been selected for viewing. For each of the datasets that the user selects to view they must also supply an appropriate identification number for that dataset. This identification number acts as a point of reference for which the spatial data viewer displays the data around. Once the user has selected the datasets that they wish to display they simply hit the "Display Data" button to display the data.

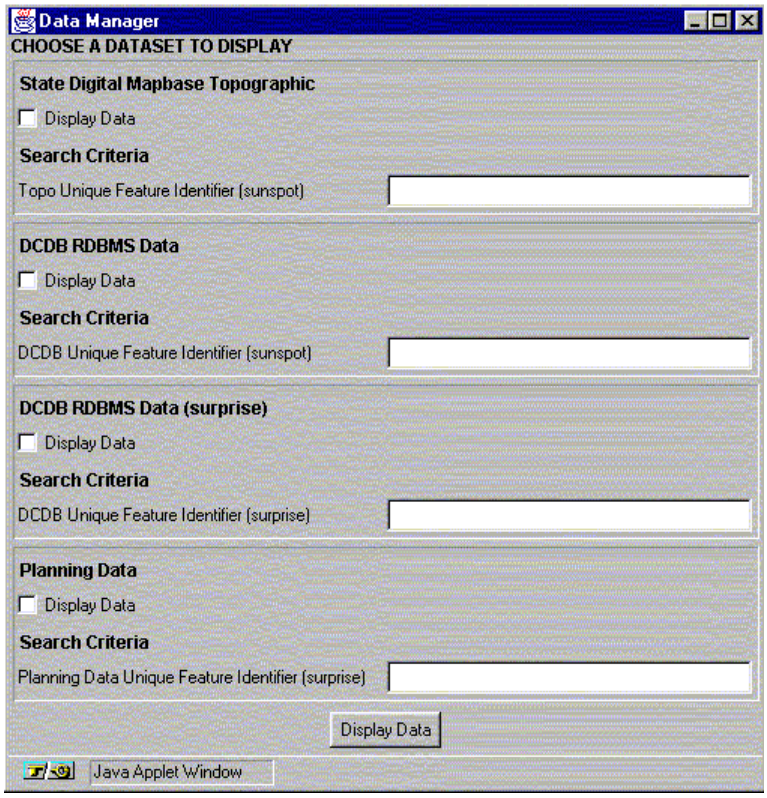

Figure 1-7: The data manager menu.

7) Once the "Display Data" button has been depressed the spatial data viewer then consults the "Directory_Andrew.txt" file to find the access metadata required to format and send a query to each of the remote/local database systems. The query requests from the database all the spatial data for 100 metres, centered around identification number supplied by the user. The results from each of the queries are returned to the spatial data viewer by the remote/local databases systems where they are displayed.

The spatial data viewer that was developed has the ability to zoom in and out, pan, fit and clear the data. The viewer does not have the ability to query each of the datasets to a more complex level than simply requesting data centered around a specific point.

1.7 Modifications made to Isite to develop the Prototype

The software package Isite was used as a base for the development of the metadata engine prototype. In order to get the Isite software to act as a metadata engine, rather than just a metadata system, three stages of modifications were needed. These stages were:

1. Adapting Isite to run as originally intended by CNIDR on two different servers.
2. Modifying the Isite source code to return access metadata.
3. Linking the developed metadata engine to a spatial data viewer.

The rest of this section will describe in detail the modifications that were undertaken in each of these stages.

1.7.1 Adapting Isite to run as originally intended by CNIDR on two different servers

Isite is a public domain product that is available for anyone to down load over the Internet. The installation instructions inform the user that the process of installing the package is straight forward. Only a few options needing to be adjusted in the supplied Makefile to account for differing systems. After making the relevant adjustments and running the Makefile it was discovered that the installation was not so straight forward. The Isite source code had several bugs in it that prevented it from firstly compiling, and secondly running correctly when it eventually did compile. The rest of this section discusses some of the problems that were encountered in the successful installation of the Isite source code. The measures that were taken to rectify them are also discussed.

1.7.1.1 Getting the Source Code to Work as CNIDR intended

At the time the prototype was starting to be developed the latest stable version of Isite was version 1.05. There were several untested versions of the software around,

however CNIDR could not guarantee that they would work. It was fortunate that at about this time the Prototype Distributed Data Directory was launched by ERIN in Canberra. The ERIN prototype eliminated some, but not all, of the bugs from the previously unstable version 2.06. It was decided that this version of Isite should be used to develop the prototype for two reasons:

1) It was the newest supposedly stable version of Isite available; and

2) It is the version of Isite that was being used to develop the Australian Spatial Data Directory.

The second point is pertinent as it would be useful in an Australian context to be able to extend the Australian Spatial Data Directory to encompass data access capabilities. The current plan for the Australian Spatial Data Directory is for it to be a metadata directory of all spatial datasets, no data access is planned. By using the same base package to develop the prototype the research will be able to prove if the Australian Spatial Data Directory is able to be extended.

After the Isite source code was obtained from ERIN it was discovered that there were still three bugs in the code that had to eliminated. These were:

1) A function was used in the source code that was not defined or in any of the standard libraries. After an extensive examination of the source code it was discovered that this function actually performed exactly the same task as another function that was already defined in the source code. All references to the undefined function were subsequently replaced with the equivalent reference to the function that was defined and performed the same task.

2) The Isite software allows users to search for datasets located across a distributed network of metadata databases by extracting the users search terms from a HTML search form. Originally Isite was designed to perform one of three different types of searches using a HTML form. Each of the searches had a varying number of search parameters that had to be filled in before the form was submitted. Once the form was submitted the Isite source code would read the parameters from the

completed form to construct the query. It was at this point that another bug was found in the Isite software.

The problem occurred when the parameters were being extracted from the HTML search form. The source code was written to read in every parameter no matter which of the three types of searches was being conducted. When the source code tried to read in a parameter that did not appear on the search form it would crash.

The bug was fixed by making each of the web pages and the code to read the parameters more generic. This was done by placing extra code in the web pages that assigned a value of "DEFAULT" to all the parameters that did not appear in the original web page. These parameters were of the type "HIDDEN" so as not to be visible to the user.

The extra code that was added to the web pages was supplemented by extra code in the Isite source code. The extra code essentially checked to see whether the parameter that was read was equal to the string "DEFAULT". If this was the case then it would set the parameter to the predefined default value that was to be assigned if the source code did not read a parameter from the original web page.

3) The Isite software allows users to interactively search distributed metadata databases by establishing z39.50 stateful gateways between the central server (where the metadata engine is located) and each of the distributed databases. The parameters that are required to establish the stateful gateway are not hard coded into the source code. Instead they are read from a file called "gateway.ini".

   The installation instructions for Isite state that the gateway.ini file should be located in the cgi-bin along with the zcon and zgate executables. The zgate executable should run in the cgi-bin directory and it is this executable that consults the "gateway.ini" file. Since the "gateway.ini" file resides in the same directory as the zgate executable zgate should be able to access the file. Unfortunately this was not the case. For some reason the zgate executable, despite residing in the cgi-bin directory did not actually run in that directory, hence it could not locate the "gateway.ini" file. To solve this problem the name of the "gateway.ini" file was

modified to include its absolute address. Hence the file was referred to as "/usr7/www/cgi-bin/gateway.ini" rather than "gateway.ini".

1.7.1.2 Modifying the gateway.ini file

As mentioned in the previous section the "gateway.ini" file is used to define the parameters required to establish a z39.50 stateful gateway between the local server and any number of remote servers. The file is set out in a certain format. To help new users create this file a template is provided with the Isite package. Obviously this template had to be modified to allow for access to the databases that the prototype wanted to access. Along with the modifications to the file to allow the correct databases to be accessed, the file also had to be modified to reflect the system architecture of the local server, rather than the architecture of the example server.

1.7.1.3 Modifying the zserver.ini file

The "zserver.ini" file is a simple text file that is read by the zserver executable to configure itself. It contains information vital to zserver concerning the location of the access log file, a list of the metadata databases located on the server, the level of debugging information to be printed to standard error during server operation, the maximum number of z39.50 connections that can be received by the server, the TCP port on which connections are to be made, the location of the "sapi.ini" file, the type of server that is to be run, and finally the number of seconds of inactivity after which the server automatically closes the session (Gamiel and Warnock 1994).

To make the installation of the Isite software easier CNIDR provides a template that is in the correct format. In order to get the prototype working this template had to be modified to reflect the way in which the prototype intended it to operate. The template also had to be shifted into the same directory as the zserver executable that uses it.

1.7.1.4 Modifying the sapi.ini file

For the prototype to function effectively it requires access to text searching facilities and database systems. As part of the Isite package, CNIDR developed a Search API (SAPI) which generalises access to arbitrary database systems via a common API.

Any application that links with the API inherits the functionality of any database that may reside behind the API.

Installation of the SAPI involves creating a single text file known as the SAPI configuration file (sapi.ini). The file describes each of the databases that are available through the API to SAPI compliant applications. Among other things the file identifies the type of databases and the location of the databases on the file system. To make the installation of the Isite software easier CNIDR provides a template "sapi.ini" file that is in the correct format. In order to get the prototype working this file had to be modified to reflect the metadata databases that the prototype intended using.

1.7.1.5 Modify the web forms

To make the installation of the Http to z39.50 gateway easier CNIDR supplied example web forms that would allow users to access the metadata databases at the University of California, the University of North Carolina, and the Library of Congress. Not only did these forms have to be modified to search the servers that the prototype used, they also had to be modified so their appearance reflected the nature of the project. Included in the modifications was the simplification of the search process. The example web search forms were quite complex and could perform complex searches that were not really required for this prototype. Hence the web pages were modified to suit the needs of the prototype. Figure 1-5, shown earlier, is an example of the new layout.

1.7.2 Modifying the Isite source code to return access metadata

The original Isite source code returned a hyperlink to the HTML metadata record. This hyperlink could be used by the system to view the metadata record. This is useful in that it allows the user to find out what spatial datasets are on the network, and then by viewing the corresponding metadata records they are able to determine whether the dataset are of use to them. The original Isite source code did not allow for the user to directly access the datasets that metadata records corresponded to.

To make the access to the actual datasets possible more information than just this hyperlink would have to be returned to the metadata engine from each of the remote servers. For the viewing of the datasets to be achieved the metadata engine has to have metadata that will tell it how to access the dataset. This access metadata for each dataset is located in a single file on each server. This file is read by the modified Isite code and the access metadata is encoded onto the return string, along with the hyperlink, for each metadata record that satisfies the users query. Each of the individual elements that are encoded onto the return string are separated by a marker which consists of three hashes ie ###. If a "Planning.html" metadata record satisfied a users query the access metadata record shown in Figure 1-4 would be consulted and the following return string would be encoded and returned to the metadata engine:

Hyperlink### Planning Data###planning###surprise.sli.unimelb.edu.au ###/opt/local/java/bin/java -classpath /opt/local/java/classes:/opt/local/java/lib/classes.jar:/opt/local/java/lib/rt.jar:/opt/local/java/lib/i18n.jar:/opt/local/java/lib/classes.zip:/opt/local/java/lib/postgresql.jar:/usr7/www/cgi-bin/ Query_JDBCjdbc:postgresql://surprise.sli.unimelb.edu.au:5432/planning### /tmp/JDBCPlanTemp###Planning Unique Feature Identifier(surprise)### Textfield###ufino

Note: This return string is all one line.

1.7.3 Linking the developed metadata engine to a spatial data viewer

In the original source code the results that were returned by the remote servers in response to a users query were simply hyperlinks to the metadata record that satisfied the query. The hyperlinks were written into a HTML page so the user could view each of the metadata records simply by clicking the relevant link.

As extra information has been added to the return string the source code that places the hyperlinks on a web page will no longer work. To solve this problem the return string has to be decoded at the central server. The decoding process breaks the return string up into the individual components that were joined to make it. The code that

decodes the string looks for the ### marker in the string and breaks the string up accordingly.

Once the return string has been decoded into its individual components the hyperlink is written to a HTML file (as it was in the original Isite source code) and the access metadata is written to a file called "Directory_Andrew.txt". This text file is used by a Spatial Data Viewer to construct a menu of the datasets that are available for viewing.

The prototype metadata engine was then linked to a Spatial Data Viewer, developed by (Polley 1998), by placing extra code in the Isite source code that places a button on the Results web form that starts a Spatial Data Viewer applet. Once the applet is started it uses a CGI script called "MetaDataEngine.cgi" which creates an Available Datasets menu by reading the metadata that was written to the "Directory_Andrew.txt" file. The user then has the ability to view the datasets concurrently selecting the datasets they wish to view from the newly created menu and entering a reference point for each dataset. All lines that start within 100 metres of the point of interest for each of the datasets are then drawn onto the screen by the Spatial Data Viewer.

1.8 Chapter Summary

This chapter outlines both the choices that were made in developing the prototype and the modifications that were necessary to the Isite source code in order to allow it to act as a metadata engine. Whereas the previous chapter outlined what the options were for developing a metadata engine, this chapter discussed which of the options were actually adopted and why.

Of the data models that were described in chapter four, option 5 was chosen to implement the metadata engine. In Option 5 the metadata engine was located on a central server, and the metadata located on the data custodian's servers, or on the central server. There are several reasons as to why this model was adopted:

1) There is a minimal wastage of resources. In several of other options metadata was duplicated on two or more machines.

2) Data custodians remain in control of their metadata. This not only helps to ease the minds of data custodians it also ensures that the metadata that the engine uses should be up to date.

3) Adding new servers to the metadata engine is straight forward. In the situation where there are many copies of the metadata engine located in many, sometimes unknown, locations, updating the metadata engine to search a new server for metadata is a major problem.

4) The metadata custodian has the opportunity to have their metadata stored on the central server if they wish. This is useful where the metadata custodian does not have the ability to serve the metadata to the system themselves.

Of the two options that were outlined in chapter four detailing how to handle the access metadata it was the second option, the separate file approach, that was chosen as the appropriate method. The lack of a standard format for the lower pages in the ANZLIC metadata guidelines and the ease of implementation of the second option were two of the reasons for undertaking this approach.

The separate file that was developed to hold all the access metadata was called “directory.txt”. It is located in the same directory as the zserver executable and is set up using a standard format that has a marker before each of the entries to identify it. When a metadata record is found as a result of a user's search the metadata file is consulted to find the access metadata.

The prototype was developed by modifying the software package called Isite. Isite is a public domain package developed in the USA that uses the z39.50 ANSI/NSIO standard communications tools to access metadata databases across a network. It allows for a web based interface to be developed that can search multiple metadata databases located in multiple locations across a network.

The prototype metadata engine works in the following fashion:

1) The user establishes a http to z39.50 stateful gateway between their machine and the server where the metadata engine is located.

2) The user enters the search parameters into the search page that appears as a result of establishing the http to z39.50 stateful gateway.

3) Zserver sends the search parameters to each of the remote metadata databases where the metadata records are searched to see if they satisfy the parameters.

4) For all metadata records that satisfy the search parameters a hyperlink to the metadata record and all the access metadata is encoded onto a return string and sent back to the metadata engine.

5) Upon receiving the return strings from the remote servers the metadata engine decodes them and places the decoded information in the relevant files and web pages.

6) The metadata records can now be viewed by clicking the relevant hyperlink on the results web page and the datasets can be viewed using the spatial data viewer that the web page has loaded.

There were many modifications that were made to the Isite source code to enable it to act as a metadata engine. These modifications can be classified into three stages:

1) Adapting the Isite source code to run as originally intended by CNIDR on two different servers.

2) Modifying the Isite source code to return the access metadata.

3) Linking the developed metadata engine to a spatial data viewer.

1.