An Information Architecture for Sharing and Aggregating Geospatial Content

Brian Kernighan, Advisor

This thesis represents my own work in
accordance with University regulations.

Chris Karr, Author

"A computer terminal is not some clunky old television with a typewriter in front of it. It is an interface where the body and mind can connect with the universe and move bits about."

- Douglas Adams

"Should we throw another human wave of structural engineers at stabilizing the Leaning Tower of Pisa, or should we just let the damn thing fall over and build a tower that doesn't suck?"

- Neal Stephenson

"Would you tell me, please, which way I ought to go from here?"

"That depends a good deal on where you want to get to," said the Cat.

-Lewis Carroll

Table of Contents

Chapter 1:

Introduction

Geographic information systems (GIS) are electronic systems for collecting, storing, and applying geographic data. GIS systems[1] are presently used for a variety of tasks, including mapmaking and urban planning. These systems are used extensively in government offices, in private companies, and by individuals. Despite the breadth of useful applications that GIS systems serve, current applications only take advantage a small fraction of the potential they offer. This lack of utilization of potential is a result of many factors, not limited to various incompatible systems, the lack of a comprehensive and consistent geographic data space, and the difficulty of collecting data in the real world then importing it into a GIS system.

While current GIS technology consists of a patchwork of independent data spaces and a relatively small number of GIS vendors, this technology has the potential to alter the way people work and live. With a little imagination, it is not hard to envision a few potential applications: self-driving cars that are aware of the road conditions at all times, miniature location tracking devices that parents use to keep track of their children, and electronic agent-based automatic delivery services. However, before these applications can become a reality, a technological foundation must be built from which these applications may emerge.

The largest obstacle to this vision is the lack of a comprehensive and open geographical data space. Currently, geographical data spaces consist of private collections of geographic data in many different formats held by parties that have different expectations and plans for the data. In an ideal future, there would be a single publicly accessible geographic data space where private data collections contributed to the larger public data space. This cooperation and sharing of data would act to towards the benefit of both the collector and the public. Rather than “islands” of geographic data in different systems and formats, a comprehensive and consistent geographic data space would consist of data in an open and widely available format with open protocols unencumbered by proprietary standards and licensing. The open data and protocol formats would not be controlled by a single entity, public or private. Rather, consensus regarding the protocols and formats would be a combination of technical merit and providing for public accessibility. Furthermore, the physical data storage would not consist of a single monolithic server; rather it would consist of a network of servers where each server contained a small amount of specific data that was maintained by the party responsible for the data. While the data would reside on many different servers, the underlying protocols and formats would make the task of retrieving data and combining data from multiple servers convenient and accessible.

The applications that would use this geographic data space would be as varied as the geographic data space itself. It is easy to imagine the traditional mapping applications that retrieve spatial data and render it in a visual format suitable for humans. However, the real potential for fully using this geographic data space lies within the applications that are not directly concerned with immediate human factors (such as usability or visualization), but serve as building blocks for more advanced applications and devices.

The Purpose of This Project

This project has several goals. The primary purpose is to explore and document the technical landscape that has traditionally been the realm of commercial GIS vendors and GIS technology integrators. This project seeks to obtain this knowledge and perspective by engaging in the design and implementation of a prototype open GIS[2] system that achieves some of the conceptual and technical goals of the Digital Earth vision. This is accomplished by implementing complementary components such as a geographic feature server, a range of clients that use the server in different capacities, and establishing underlying protocols and file formats that guarantee that both the clients and servers can communicate with a minimum of interference or ambiguity.

In addition to exploring the GIS landscape, this project also seeks to make the knowledge obtained available to all parties who may have an interest in open GIS technologies. This is accomplished by providing the source code of all relevant components as concrete implementations of the ideas discussed within this thesis. In order to provide for the maximum availability of this code, and to allow others to use and modify the implementations contained within, technology in this project is based upon open and freely available technologies when possible. All of the source code of this project's implementations is made available in the accompanying CD-ROM, along with the software packages used during the course of this project.

In addition to providing the perspective and experiences of implementing an open GIS system and providing the implementation code to the public, this project also seeks to expand the collective imagination of GIS users and providers by introducing client applications that illustrate novel and innovative ways to use the power of this platform to accomplish goals that are beyond the scope of traditional mapmaking and route planning packages. Hopefully, some of the applications described and implemented within will provide a spark of imagination that others may use to develop and implement applications that creatively use this and other open GIS platforms.

Finally, this project humbly hopes to seed some imaginative thinking and application design by applying all three of the previous goals. This project's goals will be fully realized if some developer finds the information within useful and is able to use or adapt some of the included implementations to accomplish something interesting. By releasing these components freely, this project seeks to reinforce the social norms that encourage the open development and cooperation within a developer community. Since much of this work was built upon the efforts of others who had shared their efforts and ideas, it is only

fair that this work is similarly released so that others who find this work valuable may be enticed to continue this tradition.

Project Implementation

This project takes advantage of several existing technologies. These technologies were chosen because they were widely available and supported rapid development and prototyping.

When possible, the Java programming language[3] has been used as the implementation language for components of this project. With the exception of the PocketPC client component of the KeyHole application, all components were created using Java. Java provided a clear advantage over other technologies because Sun and other vendors make robust Java software development kits available for no charge. Java is also widely supported in the developer community and documentation about various facets of the language is readily available online and in books. Furthermore, for some of the technologies that are building blocks of this project, the reference implementations are available in Java and the Java counterparts are generally more widely supported than the alternatives. This was an important factor when integrating XML technology because the XSLT, SVG, and parser technologies are very well supported in the Java community. Java was chosen as it provides for the design and implementation of complex software that is often more difficult in other languages.

Throughout this project's components, Scalable Vector Graphics (SVG)[4] technology was chosen as the foundation which visualization components were built upon. SVG was a natural choice for this role as the geographic data in this project is primarily vector-based, and SVG is a format that is designed for vector applications. Furthermore, SVG is well supported as a both an image file format and as a Java Swing applet when using Apache's Batik toolkit[5]. Because of this convenient duality, it was less programming intensive to use SVG to natively display vector content.

SVG is one instance of XML technology that was used in this project. Other XML technology was used extensively in varying capacities. The communications protocol was written as XML-based exchanges. The common language for describing geographic features is an XML derivative called Geography Markup Language (GML)[6]. XML provided a common foundation that supported operations such as translating one data format to another, and XML provided a convenient programming interface for storing information within applications using the Document Object Model (DOM) Java libraries.

These technologies are used in the implementation of the client-server model described below.

Chapter 2:

Architecture and Basic Server

At the heart of any geographic information-sharing arrangement, the central component is a server that clients query for geographic feature data. The architecture developed by this project is no exception.

The architecture used is the client-server model where the intelligence of the system resides primarily at the server and client ends. This architecture is very similar to the ones used with relational database models, where the client is assumed to be intelligent enough to generate the exact queries that it wants fulfilled and the server is intelligent enough to fulfill those queries. The architecture and components described below can be thought of as extensions to this model where the client application acts as the querying agent and the server acts as the database fulfilling queries.

However, despite the similarities between this geographic data architecture and traditional databases, there are a number of important differences that cannot be ignored. First of all, where traditional databases are designed to be a general-purpose tool for storing and serving many different types of data, the server in this architecture is primarily focused towards data with strong geographical and geometrical components. This leads to certain types of queries that have no equivalent in the traditional database models such as queries that utilize the relationships between two and three-dimensional spatial components. Also, differences start to become clear as different types of geographic features are added to the server's collection. Where in traditional databases, the types of data that were part of a database were heavily regulated by the database table schemas, in this data architecture, the requirement for stringent structure requirements is loosened to accommodate the property that the definition of an object such as a highway may be drastically different than the definition of a building or river. This data architecture must be flexible enough to accommodate many types of feature definitions. This is not to say that there may be no consistency between any two features (this may be enforced within classes of features using certain schema definitions detailed below); rather it is meant to imply that a certain amount of flexibility is required in order to account for the diversity of real-world geographic feature types.

Perhaps the best way to identify and define a server is to define the operations that a server must be able to execute. Within this architecture, the server must be able to execute the following operations:

- The server must respond to a class of queries originating from a client that will allow the client to query the server for specific geographic features ("Where is

Main Street?"), to query for certain classes of geographic features ("Where are all the interstate highways?"), and to query for geographic features that share common location attributes ("What are all features that are contained between 27.5° N, 130° W and 27.75° N, 131.25° W?"). Furthermore, since the amount of geographic data may be overwhelming for a potential client, the server must also provide filtering services that allow the client to conveniently choose a subset of features from the queries above by specifying the desired types ("What are all parks, streets, and airports contained within 27.5° N, 130° W and 27.75° N, 131.25° W?").

- The server must allow clients to contribute to its geographic feature collection by providing mechanisms that allow the client to define a geographic feature that is to be included in the server's feature collection ("Please add a feature of type 'street' to the collection with the attributes A, B and C."). Furthermore, the server must provide the mechanisms for clients to modify and delete features within the server's feature collection.

- Since the operations above have the potential to be potentially damaging and resource intensive, the server should implement an access control framework that allows administrators of the server and its feature collection the ability to restrict which clients may specify certain operations. This framework is intended to provide for relatively fine-grained access control and should be aware of different client types (similar to user agents in web browsers), different users behind a client (similar to Unix-style users and groups), and the network location of a client (client's IP address, hostname, and network). Using these three qualifiers, a server's administrator should be able to craft an access policy that is implemented by the server in order to allow and restrict access to features and operations as is necessary.

Since the server is delegated the responsibility for storing and providing access to the geographic data and related operations, the client software has the responsibility of using the data obtained from the server in a useful manner. Potential applications may include generating maps from the feature data, providing a user-friendly interface to the server and accessing and manipulating its data (given the proper permissions), or converting the data into a different format that other applications may use. The number and types of potential applications is only limited by one's imagination, though several will be described later in this thesis.

Implementation

To implement and deploy a server based upon the requirements listed above, a number of items must be considered so that the appropriate underlying technologies are used to construct the server. Given the descriptions above, a number of requirements are evident.

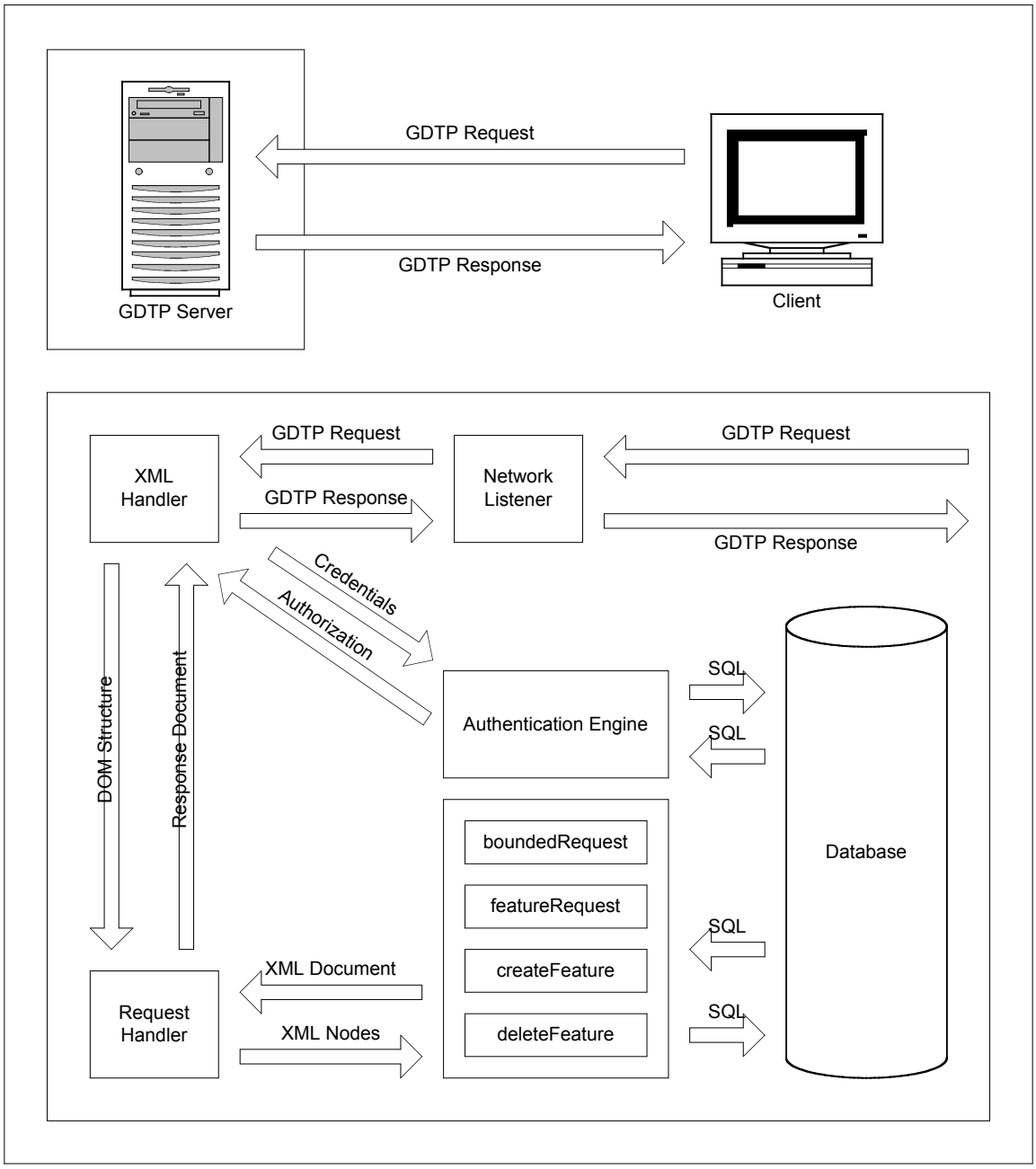

Figure 2.1: From the perspective of a client application, all interactions with the server happen on a request-response basis. Within the server, a number of steps are required in order to fulfill the request. A request must be parsed into the necessary DOM data structures, the DOM structure must be parsed, and parameters for requests are obtained from the DOM structures. These parameters are provided to the appropriate request handler that interacts with the database to fulfill the request.

1. To facilitate communications from a wide variety of clients, the server must be accessible outside the physical host on which it is running. This condition suggests that the server must be networked and be able to communicate over existing networks. Given the overwhelming adoption of the Internet and related technologies, the server should be able to communicate via this network using standard network conventions.

2. The range of client applications extends from small, embedded tracking devices as described below to large applications involved in the analysis of data obtained from the server. Thus, no assumption about the types of hardware or software can be made, and the server should attempt to communicate with as wide a variety of applications and platforms as possible.

From the perspective of this project, a number of optional items are also desired.

1. The server should incorporate existing commodity technology where appropriate in order to minimize implementation time. Furthermore, any technology utilized should be freely available (no cost and no burdening intellectual property constraints) as to not erect unnecessary barriers to development.

2. The server should be hardware and operating system independent in order to be able to adapt to changing hardware and software platforms and to take advantage of trends such as changing hardware and software platforms.

3. Installation and configuration of the server should be relatively simple and not require the server administrator to master an obscure command language. Furthermore, operations such as managing users and permissions should be easily accessible from whatever interface is provided for configuration and management purposes.

Given these requirements and optional features, this server is designed to operate from a Java-based platform, utilizing a MySQL database[7] for data storage and internal queries, with XML being the basic language in which queries, responses, and feature data definitions are encoded.

MySQL is used for internal data storage as it provides a straightforward interface for accessing data (SQL) and an efficient and robust engine for manipulating and retrieving data. While it may seem counterintuitive that a standard relational database is used while the server must be more general, an internal solution was developed where the flexibility of data types can be translated into standard MySQL tables and rows. Furthermore, MySQL offered an advantage in that it is available freely online and includes the optional InnoDB[8] backend, which was used extensively within the server for better performance. This is described in more detail below.

XML is an attractive technology, as it is a standard platform-agnostic encoding mechanism that is capable of encoding complex objects in a manner that is intelligible to both man and machine. The XML derivative GML is used as the primary method for encoding feature data. An original protocol called Geographic Data Transport Protocol (GDTP) was developed in order to encode communications between clients and servers and to encapsulate GML data. Using the web as an analogy, GDTP is to HTTP as GML is to HTML.

Finally, in addition to the technologies above used in the core server component, an optional web-based configuration component is included in order to provide a convenient and straightforward interface for configuring and managing the server. This is accomplished by including a simple Java-based web server that has access to configuration properties and runs in parallel to the main server. This is described below in more detail.

Server Overview

The design of the main server is such that it can be subdivided conceptually into two separate servers[9]. The main functionality is contained within the GDTP server. This component is responsible for accepting GDTP queries and returning appropriate responses. Parallel to the GDTP server is an HTTP server that provides an HTML interface for configuring and managing the server[10]. The bulk of the functionality is contained within the GDTP server.

The GDTP server consists of a multithreaded server that spawns a new thread in order to handle incoming requests. The server listens on a dedicated port (9000 by default), and once an incoming request is received, it spawns a new thread to handle the request. The incoming request is parsed and passed to an XML handler that is responsible for verifying that the request is in proper form. If the request is found to be in good form, it is translated into an XML DOM tree, which is passed to a GDTP request handler[11].

The bulk of the functionality is contained within the GDTP request handler. The request handler parses the request's XML DOM tree and executes the query contained within. In cases where feature data is required, this component communicates with the external MySQL database to obtain the requested feature data. This data is parsed and inserted into another DOM tree that represents the eventual response that will be returned to the client. Once the necessary operations have been carried out and the response DOM tree generated, the GDTP handler translates the DOM tree into textual XML data encoded in ASCII that is returned to the calling XML handler. The XML handler returns the XML document to the threaded network layer, which transmits the response to the client. The connection is closed and the thread is killed.

In addition to the socket interface provided to applications, an administrative web interface is included in order to assist server operators with the configuration and management of the server. Access to the administrative functions is limited to

administrative users. Upon initial installation of the server, a default administrative user is created with a default password that can be used to set up the server and delegate administrative responsibilities amongst other users. There are three main tasks that administrators can engage in: create new user accounts and set up account information, manage feature data, and configure details such as port numbers and server names.

Regular users can also use the web-based interface, but they are limited to modifying their own account information and managing their features. Future versions may include tools that enable users to grant ownership of features to other users and apply finer grained permissions provided by the underlying architecture.

Development Server Implementation

For the development of this server, a personal computer using the Windows 2000 operating system was used. The machine used a 400 MHz Pentium II CPU with 384 MB PC100 SDRAM. Installed alongside Windows 2000 was Sun's Java2 1.4 software development kit and MySQL-Max 3.23. In order to obtain XML functionality within Java, the Apache Project's Xerces[12] toolkit was also installed. Connections to the MySQL database were established using the MM.MySQL JDBC drivers.

Since the performance of the database becomes the largest bottleneck in this system as the data collection grows, MySQL was configured to use the InnoDB backend to store and retrieve data because it is considered to be one of the fastest relational database backends available. For the purposes of developing the server, MySQL was installed with two gigabytes of dedicated disk space and two hundred megabytes of dedicated memory for the InnoDB backend. In a typical production environment, the resources dedicated to the database would be larger, with the RAM dedicated to MySQL occupying seventy to eighty percent of the available physical memory. Since the development server was to be used for other purposes, these quantities were scaled down enough to accommodate approximately fifteen to twenty counties worth of TIGER/Line[13] geographic data.

For the Java-based components, the defaults specified by Sun's Java Virtual Machine were used.

Database Architecture

The database backend supporting the server is MySQL with the InnoDB backend included. This configuration was chosen as it accommodates large tables well with better than average performance. The feature data in the database was separated between geographic and non-geographic components. The tables containing the geographic components were optimized through indexes to perform well for spatial queries, and tables containing the related data required to reconstruct the XML data structures were kept in other tables that contained the information required to build trees from bottom-up and top-down approaches. Furthermore, server configuration data is contained in the

database. Maintaining the configuration data with the database allows server administrators the ability to update and apply changes in the server configuration without having to restart the server in most cases. In addition, information that is used to implement the access control framework is also within the database, including user information and host and network access tables. In short, the GDTP server is heavily reliant upon the database for much of the information it uses during the course of operation.

While a special-purpose spatial database could have been chosen as the backend for the computationally intensive spatial queries, a general relational database was used because the MySQL relational database offered many advantages while not incurring many disadvantages. In future implementations of this server, as spatial queries become more complex and demanding, it will become necessary to switch to a spatial database in order to accommodate the queries. This will be discussed in more detail below, along with a possible migration path from storing geographic data in traditional relational databases to storing the data in special purpose spatial databases. However, MySQL and InnoDB were sufficient for the purposes and scope of this project.

The combination of MySQL and InnoDB in this project was chosen because the combination offers many advantages above other freely available database systems. MySQL was chosen initially because it is relatively simple to set up on different platforms and its performance is excellent. Since this project did not require some of the more advanced features of modern databases such as transactions and versioning, MySQL's tradeoff of these features in order to achieve superior performance did not adversely affect this project in any way. Furthermore, with its integration with InnoDB, no alternatives provided the performance and ease of use and deployment offered by MySQL.

InnoDB tables were used instead of the default MyISAM tables because they offer greater performance than the MyISAM tables when databases become larger and larger. Since the number of features in the server is in of the order of millions, it was expected that the InnoDB backend would handle these queries better than the default backend. Benchmarking and timing the queries against both backends revealed that InnoDB was several orders of magnitude faster than MyISAM for datasets containing as many as fifteen to twenty counties worth of data, consisting of approximately one million points per county.

The collection of the tables contained within the database consists of several operational data tables and two main content tables. The operational data tables contain configuration information, user definitions, group memberships, permissions, host access tables, and network access tables. This information is used to provide for functionality such as access control lists, associating features with users, allowing users to form groups in order to

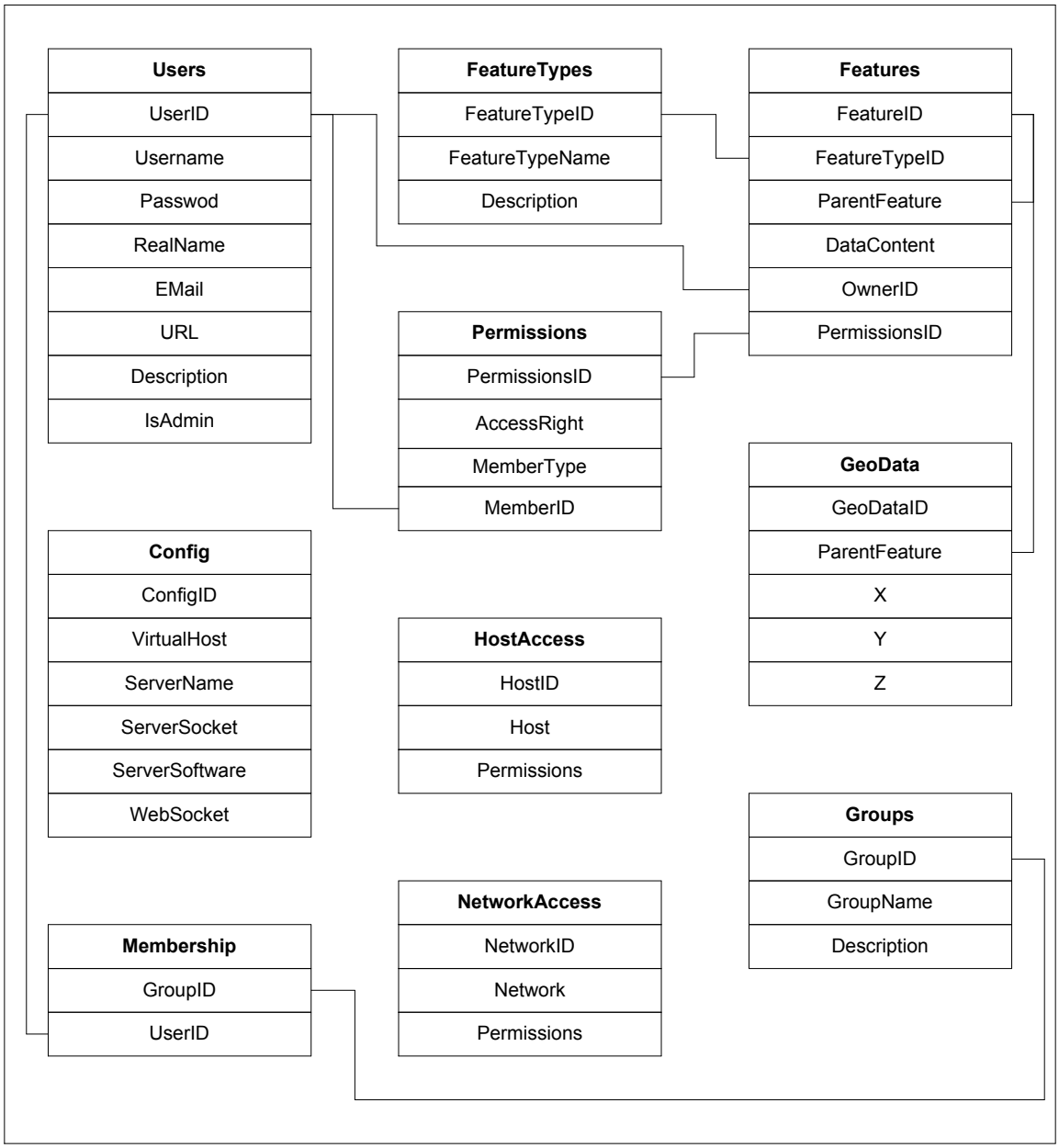

Figure 2.2: This diagram illustrates the table schemas and the relationships between tables. Bold text on a grey background represents the name of the table, and lines connecting individual components represent foreign key relationships between separate tables.

share data, and information about what hosts and networks are allowed or denied access to the server.

The configuration table consists of rows that represent different virtual hosts served by the GDTP server. Since the virtual hosting functions are not fully developed at this point, this table consists of a single row. Within this row, the server name, server socket, server software, and web sockets are set. The server name is used to provide a descriptive identifier to clients. For example, a test server could have the name "Development Server, Version 0.57" or something similar. The server socket parameter determines on which TCP port the server accepts GDTP requests. The server software is used to provide software information such as software name and version. Finally, the web socket column determines which TCP port is used by the web interface.

The user definition table strongly mirrors the idea and implementation of the standard Unix *passwd* file. Each row in the table corresponds to an individual user. The required fields for each user are a unique user identifier, a username, a password, and whether a user may access the administrative functions of the server. Optional fields are the user's real name, an e-mail address, a URL, and a description. The information in this table is primarily used to implement a username-based access control framework where users may be granted access to restricted features by providing their username and password as credentials.

Expanding upon the implementation of users, the database also provides the tables required to associate users with groups. The main groups table consists of a collection of rows, where each row is a different group. The information associated with a specific group is a unique group identifier and name. An optional description may be provided. In addition to the main group table, a membership table shows which members are members of which groups. The membership table consists of two entries, a numeric user identifier that corresponds to a user's user identifier and a numeric group identifier that corresponds to a group identifier. If a user is a member of a group, a row in the database can be found that contains that user's unique identifier and the associated group's identifier.

A table that contains the host access information is also within the database. This implementation is based upon the Unix *hosts.allow* and *hosts.deny* files that specified host access permissions through hostnames and IP addresses. In this project, the host access table serves the same purpose. The host access table always has at least one entry that specifies the default access permission: allow or deny. Additional entries containing a hostname or IP address may be added to the table in order to create exceptions to the default rule. Thus, if the default access is allow, to deny a host access to the server, a deny entry must be made in this table. Conversely, if the default access is deny, entries must be added to the table granting access to the server from other hosts. A similar table exists for networks, where a network can be specified by domain name or IP address range. The process for allowing and denying access is identical to the process used in the host access table.

The permissions table is used to set permissions upon different features. The Unix security model has heavily influenced the model within the server. Access rights can be set that allow users to grant or deny access to specific users, groups, or general users. Currently, the permissions framework is not fully developed and this table is provided for the future implementation of a more robust security model.

While the tables described above contain meta-data for server operation and tasks, the tables below contain the bulk of the content that the server manages. The first content table is the feature types table. This table is used as a catalog of features that the server is aware of and features that are present within the database. Each feature type has a unique numerical identifier, a name, and an optional description. By default, there is always an entry for the *"gml:coordinates"* type, as that feature type is the one used to store geographic data points. Other feature types may be added as needed. However, for all features within the database, an associated feature type must be present for the feature and all sub-features.

One of the two main tables for storing content is the features table. The features table contains all of the non-geographic information that is associated with geographic features. This includes information such as feature names, descriptions, and XML composition. This table was designed so that the tree structure of XML documents could be preserved in a flat table space. This is accomplished using a recursion-based implementation of the feature table.

Each feature has a unique numerical feature identifier, a feature type identifier, a parent feature identifier, a data content field, an owner identifier, and a permissions identifier. The feature identifier is a unique integer used to reference a particular instance of a feature in the database. A feature instance may be one of two types: an element node containing sub-features of different types, or a content node that has no child features. With the exception of feature trees that terminate with a *"gml:coordinates"* node, all terminating leaves of a feature tree are content nodes. For example, a node that represented the name of a feature would consist of a feature type that corresponded to a *"gml:name"* type in the feature types table, a parent feature identifier that contained the parent of the current node, and the actual name of the feature stored in the data content field. Within the table, there would be no nodes that referred to the name node as a parent node. However, if the node is a parent node, the data content field would be empty, and one or more nodes would refer to the current node as a parent node. This structure allows for XML data to be translated into flat tables, and using a series of SQL select queries, the server can rebuild the original XML structure.

This model changes slightly once geographic data is being accessed. Where all data for all nodes with the exception of *"gml:coordinates"* is contained in the features table, all data associated with *gml:coordinates* nodes are contained in a geo-data table. This table consists of a collection of rows representing geographic points composed of a unique identifier for a single point in space, a parent feature that the point belongs to, and X, Y, and Z coordinates of the point. All points that belong to a single *gml:coordinates* node have the same parent identifier.

The reason for the separation of these content types into two different tables is because of different types of queries that use the data differently. Since this is a geographic database, queries of features contained within a bounded area are common for mapping and other purposes. Since the data that is evaluated in this context is the geographic data, implementation is simpler by maintaining this data separately from the other XML content. Furthermore, large increases in performance are realized as the database can index the content along the X, Y, and Z-axes.

This design works well for purposes intended, but there are still a few problems. The primary problem is that as the number of features in the database grows, the access time is degraded. This degradation is amplified in insert operations after a database is indexed, as the database must rebuild the indexes for each insert. Furthermore, as the database grows, a ramping-up period is required to allow the database to swap in data from the permanent storage into physical memory. This effect is most noticeable when the database has been primarily querying a specific geographic locale, and then is immediately asked to start working on a different locale. This behavior was noticed when querying a database that was about two gigabytes in size, including tables and index structures, on a computer that limited the database to two hundred megabytes of dedicated physical memory.

While some problems persist, a number of solutions are available to deal with these problems. These solutions are discussed in more detail below. Despite these problems, this database architecture is very responsive for smaller to medium data sets, and it provides a good general interface for the Java components of the GDTP server to query and manipulate features efficiently. Also, by separating the general XML content from the geographic-specific content, this architecture is able to incrementally adapt to emerging spatial database technologies by allowing these new technologies to substitute for the geographic components of the system. This is also discussed in more detail below.

Geographic Data Transfer Protocol (GDTP)

The Geographic Data Transfer Protocol (GDTP) is an application-layer protocol that allows the GDTP server to communicate with client applications. Its design is heavily influenced by HTTP, though some differences become evident when it is used for spatial queries.

A GDTP session consists of a request sent to the server from a client, and a response returned by the server. The protocol is XML-based and all GDTP communications are encoded as an XML document. The typical structure of a request document consists of two parts, a metadata part and an actual request. The metadata component of the request contains information such as user agents, username / password credentials, and other information that is useful to the server, but not central to the request. The actual request can consist of a combination of any of the following request types: create a feature, request information on a certain feature, request information on all features contained within a certain bounded area, delete a feature, modify a feature, search for a feature

conforming to a certain search criteria, and request server information. For requests that are inherently read-only, such as request a single feature or a geographic area request, username and password credentials are optional depending upon the access permissions of the features identified by the request. For requests that require write access, such as creating or deleting a feature, supplying credentials is mandatory since such operations usually require privileged access.

A response returned by the server consists of a status code that identifies the success or failure of an operation, and the data requested. Since the geographic data contained in a GDTP response is GML, a GDTP response document encapsulates a GML document containing the results of the request.

GDTP was designed as an XML-based protocol for a variety of reasons. Using XML as the base of the protocol reduces the number of errors that can be introduced by building a request or response document when applications apply XML parsers. Furthermore, since the data being requested is returned in XML, it made little sense to use a non-XML text or binary protocol to encapsulate the information. On both the client and server side, XML is easier to parse using XML toolkits than creating custom string parsers that each application would implement. Finally, XML provides a strong advantage since GDTP documents are XML documents, so they can be parsed into other document types simply by using an XSLT[14] parser and stylesheet. This allows applications to easily convert GDTP output into formats such as PDF and SVG.

CreateFeature Query

The *createFeature* query type in GDTP is used by applications that submit geographic content to the GDTP server. Typically, a *createFeature* request contains username and password credentials to identify the owner of the new feature, and embedded in the request is the GML definition of the feature itself.

The server uses a simple process to insert the new feature into the database. Once the server receives a GDTP request, the request is translated into an XML DOM object. The server finds the DOM node that corresponds to the root of the encapsulated GML document and passes the node structure to a function that recursively walks the tree and inserts the node and its children into the database as needed. Regular XML content is inserted into the features table, and coordinate data is inserted into the geographic data table. Upon the successful creation of the feature in the database, the server returns a success status code and the numeric identifier of the new feature.

FeatureRequest Query

The *featureRequest* query type is used by applications that allow users to select single features from the database regardless of any geographic qualities. Users may request the feature based upon the numeric feature identifier, or if the feature contains a *gml:name* component, based upon the name of the feature.

In this operation, the server requests the feature based upon the criteria given, finds the top-level parent of the feature (referred to as the root node), and builds the GML representation by recursively walking down the virtual tree structure in the database and building a real tree structure in XML. Once this process is complete, the tree representation is embedded into an existing DOM tree containing the response document, which is then converted into a text form of XML that is transmitted to the client.

The difference between requesting a data based upon a numeric identifier and a name is that in the cases with the numeric identifiers, the identifiers reference the root node, and the tree is immediately built downwards from that node. If a name is used, the name node is identified first, then the root node is found, then the tree is built downwards. While this is less efficient, the performance penalty of finding the root node is generally small enough to justify the inclusion of this operation for the users' convenience. Also, querying features by name can return multiple features with the same name. If a query requests a feature called "Main Street", all "Main Streets" in the collection will be returned.

BoundedRequest Query

The *boundedRequest* query type is probably the most often used. It permits an application to search for a collection of features that is within a certain geographic area. When combined with filters, this query type provides users with a simple and powerful tool for obtaining location data.

A bounded request consists of a mandatory *gml:Box* element that outlines the rectangular area that is to be searched. In addition to the *gml:Box*, applications may provide a filter specification that limits the types of features returned and limits the work to those feature types, providing a significant improvement in performance for large areas with many feature types.

Since this query type queries more features than any of the other types, bounded request queries are most sensitive to database performance and other factors. In the database design described above, the majority of indexing and structuring decisions were made to optimize the performance of this operation.

More steps are required to complete this operation than the other operations. Upon receiving the request, the request is parsed into a DOM structure that is used for navigation. The *gml:Box* element is located and the range of the X and Y coordinates is

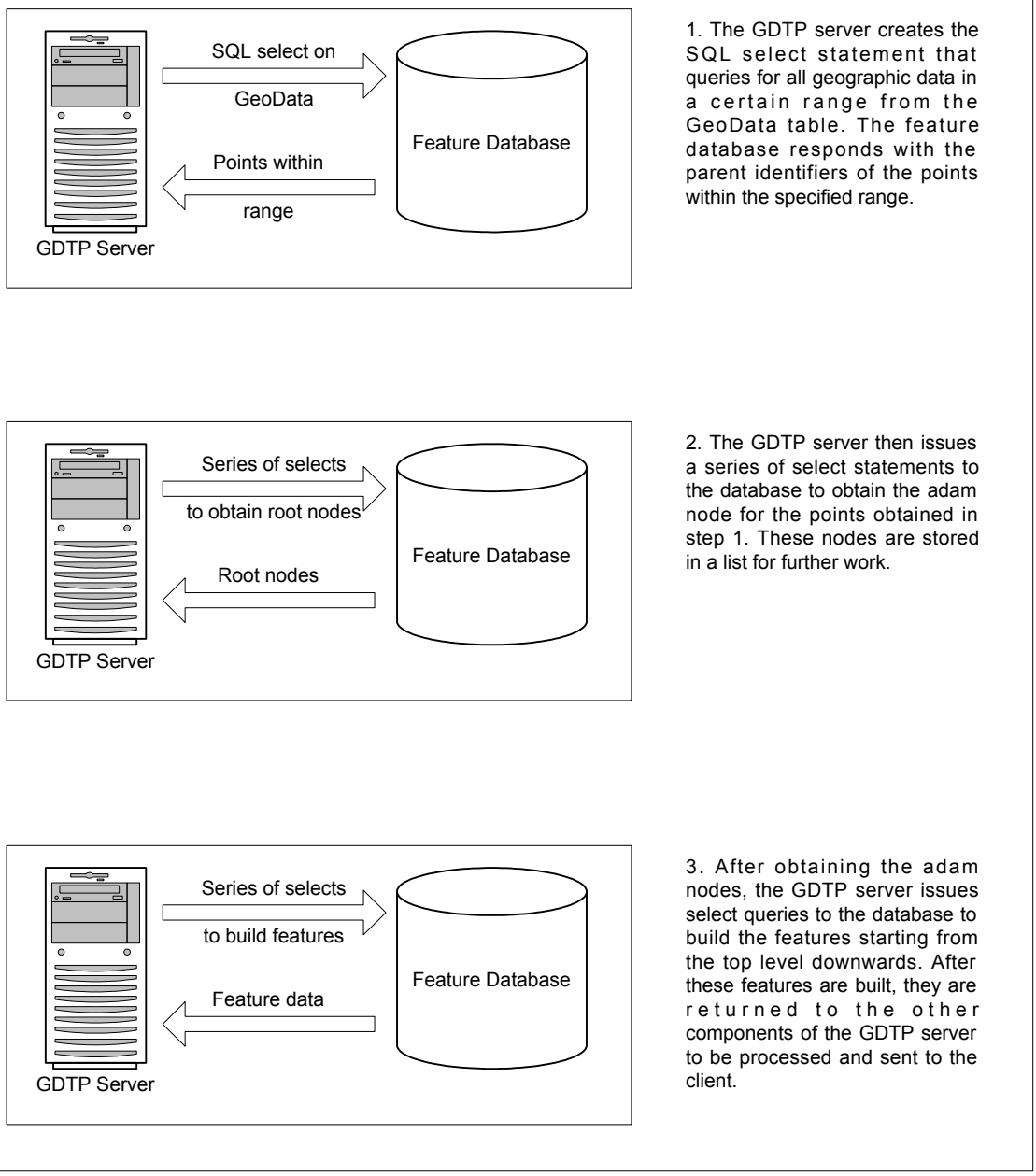

Figure 2.3: The communications between the GDTP server and the feature database are required to complete a boundedRequest.

extracted. After extracting these parameters, a SQL query selects all parent identifiers of the coordinates in the geographic data that are within the bounded area. After the parents of the points are located, the filter is applied (if present) to select the types desired. After this list is built, the node features are found, and then the features are built downwards (as in the *featureRequest*) and included in a new DOM structure. Once the DOM structure is complete, it is merged with the existing response DOM structure, converted into textual XML, and then returned to the client.

A number of design decisions were made in order to balance programmability and performance in this query. The filter, as implemented, operates on the Java side of the server, discarding feature types that were not requested. It could have been just as easily implemented as an extension of an SQL query, but the overhead of the joins and additional comparisons in the SQL engine would further impact the performance of the database. Since the performance bottleneck at this point is the database, the decision to burden the Java side of the server with this functionality was made to distribute the load of filtering to the less burdened side.

Another decision was whether to incorporate the existing routines that find the root features into this query. These functions were designed to work at any level of the feature tree and find the root of the tree. Performance in this phase of the process may be improved by taking advantage of the fact that the tree is walked from the bottom containing the *gml:coordinates* node to the top. This provides a ripe target for caching data. If this property could be exploited, the elimination of the duplicate SQL queries rebuilding the features would result in a significant savings. An alternative approach would be to build the tree from the bottom up, filling the side branches as necessary, thus incorporating the root feature discovery and tree building in a single step. As the implementation is now, it uses the less efficient process described above.

Currently, the bounded request only uses a rudimentary geometric operation to find features. This method excludes some features in cases where a feature may be in the bounded area, but not contain any definition points in the area. This is best imagined as executing a query that is completely inside a park, yet does not return the park because the defining points surround the requested area. One possible improvement to this request is to expand the area that is actually requested by a certain percentage, increasing the chances that some of these definition points are captured. However, this is a flawed solution because on a small enough scale, it probably will not find the desired features, and on a large enough scale, it drastically increases the amount of work required for the database and server. The best probable solution to this problem is to store the geo-data in a spatially aware database that provides the appropriate data types and geometric functions to query these data types. This also can reduce the limitation that the bounded area be rectangular. This is discussed in more detail below.

DeleteFeature Query

Compared to the *boundedFeature* type, a *deleteFeature* query requires fewer computational operations. The purpose of this query type is to delete a feature based upon its numeric identifier. To delete the feature, the appropriate credentials must be supplied. The majority of the implementation is identical to the *featureRequest* query, but instead of building the feature top down and returning it to the client, this query deletes the feature from the database top down, erasing the feature from the collection. A status code returns the final status of the operation, including the identifier of the feature deleted.

The justification for only allowing numeric identifiers to specify the feature is based on the previous observation that different features may share the same name and other attributes. Since the only data that is guaranteed to be unique to each feature is its identifier, this is the only element that can be used to delete an element without inadvertently deleting other features.

ServerInfoRequest Query

The *serverInfoRequest* query is a request that does not deal with features in any way. This query type is intended to be a tool for a server to identify itself and what capabilities it supports. This query returns information such as the server name, the GDTP server software, and additional information such as whether it supports compressed or encrypted transmissions.

This query type is primarily intended to provide information about server capabilities as incremental changes are implemented. This allows clients the ability to customize their behavior in order to adapt to a wide variety of servers that may be running different versions of the server software that may have varying levels of implementation of advanced features.

Future Work

This server is hardly complete with respect to the original feature set envisioned, and much future work could be done in the areas such as virtual hosting and a more robust permission framework.

In addition to software features, a future version of the server and GDTP protocol would benefit from the implementation of a robust *modifyRequest* request type that would allow for editing of features currently in the server's collection. Deleting a feature and recreating an almost identical feature with the desired changes currently replicates this functionality. This will suffice for some for some applications, but before serious adoption of this standard can happen, a robust implementation for modifying features must be in place.

In addition to a *modifyRequest* request type, this server would greatly benefit from the implementation of a *searchRequest* request type. The current *featureRequest* provides a limited searching function by returning features with matching *gml:name* sub-features. However, some applications will require a searching mechanism that is more flexible than the one offered by the *featureRequest* query type. Ideally, a *searchRequest* query would provide a mechanism for doing numerical comparison operations in addition to the traditional regular expression searching on text. In addition to searches on individual features, a robust *searchRequest* request type would include support for Boolean operators such as AND, OR, and NOT that would combine individual search parameters.

In order to attain the goals stated earlier about implementing a unified geographic data space, future efforts must be applied towards adapting GDTP and the server to fulfill Web Feature Server (WFS)[15] requests. WFS is a developing specification championed by the OpenGIS Consortium. OpenGIS has garnered the support of a wide variety of public and industry groups and any geographic protocol that will be adopted as a public standard is likely to be an OpenGIS protocol.

In many respects, WFS mirrors GDTP functionally. However, since this project was started before WFS was made public, it was deemed more productive to continue using GDTP instead of WFS within the server. Furthermore, since WFS is still a developing protocol, it is very much a moving target with respect to implementation of features. Rather than attempt to maintain constant compatibility with WFS, the decision to develop a parallel protocol was made in order to focus on implementing features into the protocol that were required for some of the client applications.

While the underlying architecture of the server was to maximize the number of features that could be included, the programming components used in the implementation can be adapted to achieve compatibility with WFS. Some things, such as existing request and response types, will be easily adapted to WFS, while other features such as locks and transactions will need to be implemented anew within the current server.

In addition to implementing new query types and achieving WFS compatibility, the server would benefit from further work on fundamental items such as database compatibility, security, and performance. Currently, the server is designed to work with a MySQL database. Other databases are readily available, and work towards creating a database-neutral architecture using a plug-in architecture for different databases would greatly supplement this server. This type of architecture would provide an interface where the server can use different database configurations and technologies. This plug-in architecture could enhance performance by allowing spatially enabled databases to be used in place of standard relational databases. Furthermore, the plug-in architecture could be adapted such that distributed database configurations are available to the server.

In the areas of security, work towards verifying the security and access control frameworks would benefit the server. Currently the access control frameworks are less than acceptable for use in sensitive areas, and the inclusion of security technologies within the server would boost security. Furthermore, the server could be extended to

support an additional GDTP port that would accept encrypted communications using SSL[16] technologies.

Finally, general performance enhancements such as caching and prediction algorithms can be included to adapt to changing workloads and alleviate performance issues in other areas. For example, a server that was intelligent enough to store frequently requested features could enhance performance by caching the feature and avoid requesting the feature repeatedly from the database. Other changes focused upon performance, such as predictive algorithms, could potentially improve performance and quality of service.

Chapter 3:

GarminGPSTool

GarminGPSTool is a Java Swing application that is a visualization and data access companion to Garmin's line of commodity GPS receivers[17]. This application is designed to provide an intuitive interface for downloading and visualizing native Garmin data types on a computer system. The application is capable of communicating with Garmin GPS receivers using Garmin's proprietary device protocol[18], querying the receivers and receiving and translating GPS data from a proprietary format into a GML-compliant version of XML.[19]

The user interacts with the application via standard Swing user interface components and an embedded SVG viewer. The application contains a vector-based world map that is initially rendered upon application startup. Data downloaded from the GPS receiver is overlaid onto the map in order to aid in visualization. Communications between the application and GPS unit are conducted over a standard RS-232 serial line. In addition to the data download and visualization features, this application also allows users to directly upload GPS data to GDTP servers as Garmin data types, or through translation servers that translate the Garmin data types into actual feature data. Furthermore, users may download data from these servers and store the data in the GPS unit for offline use.

The intended audience for this application is the owners and enthusiasts that use Garmin products to navigate and collect geographic information. It is intended to provide a free software solution for those who wish to access GPS data directly from their desktop or laptop computers. Through the use of networked servers, this software is also targeted towards users who desire a network-based solution for downloading and sharing GPS data.

Implementation

GarminGPSTool is built using a variety of Java2 libraries provided by various parties. The interface is designed and implemented using Sun's standard Swing GUI components. Communication between the application and the GPS device is accomplished using a combination of Sun's JavaComm API[20] and a custom network layer that serves to translate the raw byte data into usable objects. These objects are stored internally using the Apache Project's DOM-based Xerces XML engine. This engine allows the application to conveniently manipulate the data into forms that are appropriate for visualization, storing in files, and transmitting over a network. The visualization component is built using Apache's Batik SVG toolkit as a Swing widget that displays and allows users to manipulate the display of the GPS data. This component also allows users

the ability to save their waypoint data in graphical formats including SVG, PNG and JPEG. The combination of these technologies within this program supports different data formats and provides an intuitive and flexible way to interact with the downloaded data.

The Swing components of this program serve as the framework for the application. The Swing components dictate the look and feel, in addition to providing standard functionality like configuration and file-system access components. The application displays a toolbar and a map that spans the width of the application. The toolbar provides the expected functionality such as opening files, saving files, and accessing the configuration framework. In addition to the main application window, a tabbed configuration child window allows users to conveniently set configuration options including communications and network server parameters.

Embedded within the Swing framework is a JCanvas component that is responsible for displaying vector data such as the base map and downloaded GPS data. This is implemented using an SVGCanvas component that is part of the Batik toolkit. Using this component instead of writing a custom extension to the standard Swing JCanvas component provides for advanced functionality within the component, like zooming and resizing, that allows for better manipulation and visualization of the vector data. Furthermore, by using the SVGCanvas component, implementing features like saving entire maps as SVG and exporting snapshots as standard raster image files is easier because complementary Batik technologies are designed to work with the SVGCanvas components.

The internal XML engine provides SVG data to the display component. This engine is implemented using Apache Xerces technology. This layer of the application is responsible for storing downloaded data internally and exporting the data to other parts of the application in SVG format, GarminGML file format, or GDTP network format. The internal data structure used to store GPS data is an XML DOM document. Initially, the document is empty. As the user retrieves data from the GPS unit or the network, the document is populated with nodes representing the different features. Once this document is populated enough such that user desires to visualize the data or transfer it to a GPS unit, the document is translated into an appropriate format. For example if the data is being manipulated for visualization, the DOM document is processed to produce the necessary SVG elements that will be rendered by the Batik SVGCanvas component. If the data is to be transferred to a GPS unit, the data is translated into a byte stream to be sent over the RS-232 serial link. This internal implementation allows for the internal data to be translated into any number of formats using custom translation routines or through the use of commodity XML technologies such as XSLT. A feature that may be added is one where a user may specify an XSLT stylesheet that can be used to translate the internal data into arbitrary data formats specified by the user.

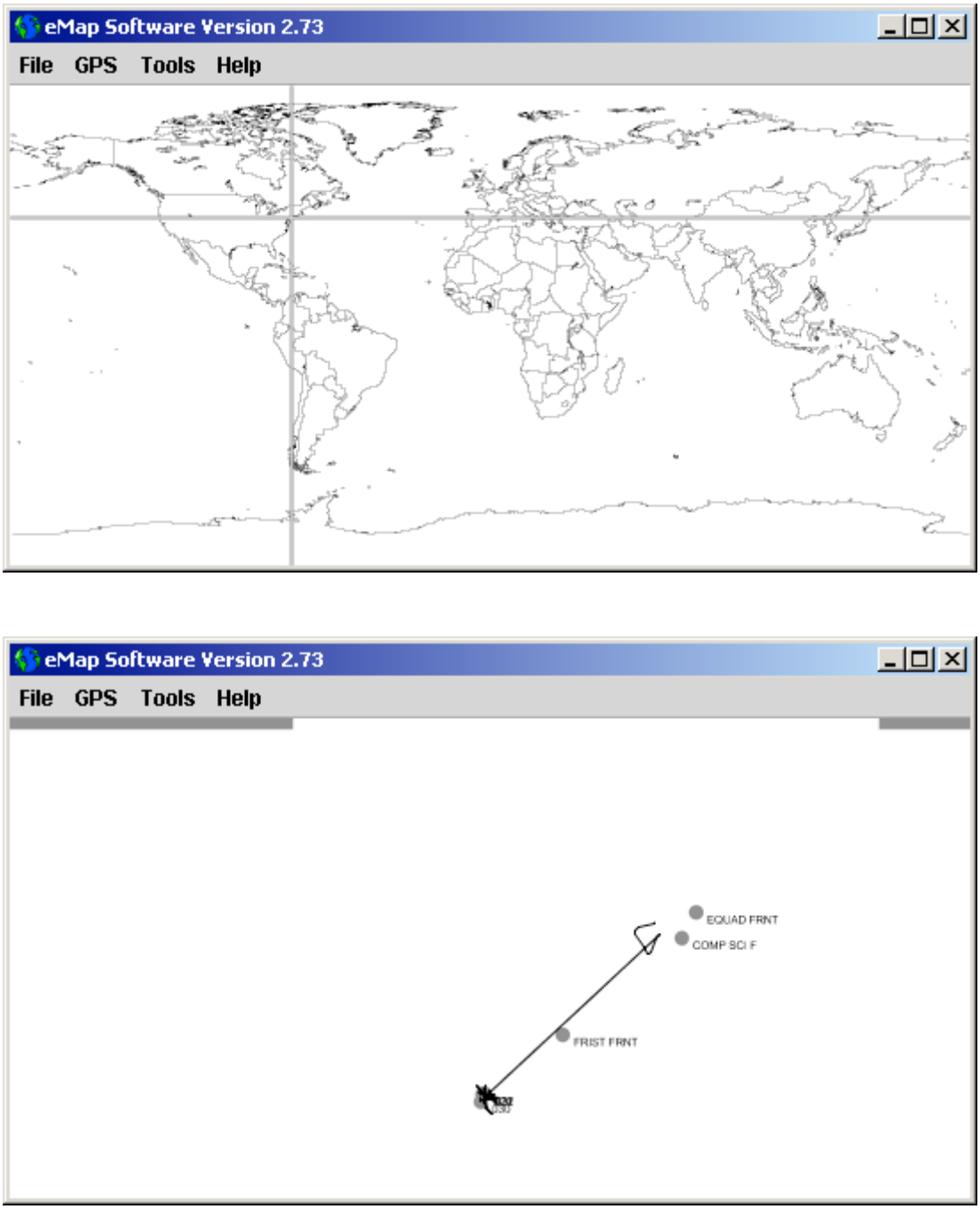

Figure 3.1: These are sample GarminGPSTool screenshots. The lines on the map on the top illustrate the approximate vicinity of the downloaded data. Users can zoom to the center of the crosshairs in order to obtain a more detailed view such as bottom image.

A serial networking component sits between the rest of the application and the Garmin GPS device. This layer is responsible for translating between the internal DOM data structure and the device. At the most basic level, communication between the application and the device is a raw byte stream that is transmitted via the serial communications port. Garmin has provided specifications that allow application developers to translate between the raw byte stream and features commands. In this implementation RS-232 serves as the physical layer and Garmin's protocol serves as the data link layer. Raw bytes from the RS-232 are translated into packets that contain byte payloads of data for the application layer. The application layer implements various functionalities including managing data types and obtaining real-time location information. A collection of data types are defined that represent command structures and feature objects. Finally, a number of application protocols dictate in what order various data types are sent and how to handle data transmissions and error correction.

Within this framework, the JavaComm libraries provide the API for reading and writing raw bytes from the RS-232 connection. On top of JavaComm, custom Java classes have been implemented that implement the link protocol and translate between packet streams into usable objects, complete with methods. Furthermore, by utilizing Garmin protocols such as the GPS capabilities functionality, this layer is able to dynamically determine which data types are appropriate for a given model line. This provides a convenient method of adding support for the entire Garmin GPS line, even though the data types supported among different models may be significantly different. Presently, data types in the eMap and 12XL models are supported.

Usage Details

The GarminGPSTool application provides custom visualization of GPS data and a simple and straightforward method for accessing and saving data. The user interface implementation maximizes functionality while providing a standard and intuitive way to access the application's functionality.

From a user's perspective, the application contains two main components upon startup. There is a standard toolbar at the top of the application, and a graphic below the toolbar that contains a world map. The toolbar contains four main items: File, GPS, Tools, and Help. The File menu contains functionality for saving and archiving data, including items for saving data to XML format or as an SVG file, and uploading data to a GDTP server. These items operate in the same manner as in the majority of GUI applications, and usage should be intuitive to users familiar with other GUI applications.

The GPS menu contains items that allow the user to interact with the GPS unit. Upon initial startup, the *Download Tracks* and *Download Waypoints* items are grayed out, leaving the *Initialize GPS* and *GPS Info* items available. The *GPS Info* item brings up a small window that allows a user to see device-specific parameters of the connected GPS unit. These parameters include the supported native data types, supported communications protocols, and software names and versions. This item is primarily

intended to be a general information tool that can communicate across the entire line of Garmin GPS receivers, and users with unsupported units can use this information to submit requests for support in the next GarminGPSTool release. The *Initialize GPS* item is used by the application to query the GPS unit and determine the data types on the connected unit so that the appropriate type-conversion methods are used in the application. Once the data types have been determined, the application is aware of enough information that it can communicate with the GPS receiver properly. From the user's perspective, the *Initialize GPS* item is used to establish communications and enable the *Download Tracks* and *Download Waypoints* items to become usable. These two items are provided so that the user may select which data types are to be downloaded into the application. However, the user does not see the new data types represented until they select *Update View* from the *Tools* menu item.

The *Tools* menu contains functionality for the user to regenerate the world map with the downloaded data types rendered upon the map. It also provides tools for resetting the view to the original world map and for bringing up a configuration dialog. If a user has downloaded tracks or waypoints, selecting the "Update View" option renders these items. The *Reset View* item does not clear the downloaded data from the map; rather it resets the original view to encompass the world map as shown after an *Update View* command. The *Options* item pops up a configuration dialog where the user may set options such as communications ports, and upload server information.

The major component of the application is the viewing area that displays the GPS data in relation to the rest of the world.

Future Work

GarminGPSTool can benefit from future work in a variety of areas. Currently, the application is limited to machines that have support for the JavaComm API and a Java2 virtual machine. At the moment, this support is primarily found in personal computers. This limits the utility of the current application, as personal computers are generally not too portable, thus functions offered by this application are lost in a portable environment. This application has the potential for larger usage if it were ported to portable platforms such as PocketPC or PalmOS. This is a straightforward improvement that mainly consists of replicating the functionality in the portable environment. Java runtime environments and libraries for the embedded platform will enable straightforward ports, or developers may port the code from Java to a platform-specific language like Embedded Visual C++.

In addition to porting the code, this application's utility can be greatly enhanced through the development and deployment of translation servers that would serve to translate the data from the GPS in vendor-specific data types to more general data types used elsewhere. Translation servers are described below, but in this instance a server that can intelligibly translate from the waypoint and track vocabulary that GPS receivers speak to

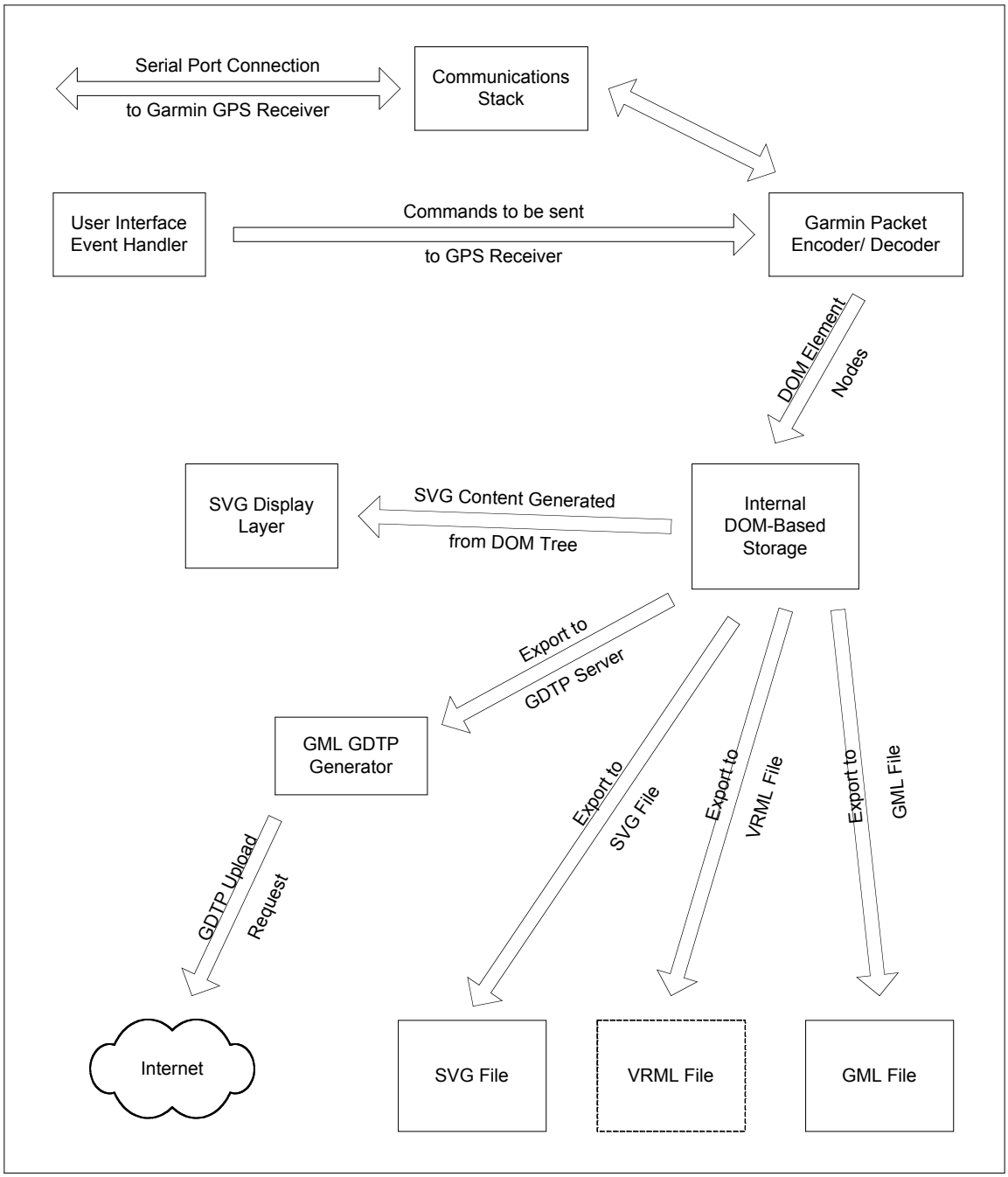

Figure 3.2: This diagram displays the major components of the GarminGPSTool application. Input events from the Swing interface are translated into commands sent to the GPS receiver and the data from the receiver is translated into XML nodes that are added to the internal DOM data structure. From the DOM structure, the SVG user interface is updated. The DOM data structure also facilitates easy encoding into other XML data file types, such as GML and SVG. The contents of the DOM tree can also be uploaded to a GDTP server. The structure makes it easy to add export support for other file types such as VRML.

a more general vocabulary spoken by other applications, such as roads and places, would greatly enhance this application's utility. Furthermore, more attention to authentication and security needs to be applied to this application. The current implementation does not account for security issues, and implementations are generally insecure.

The application can also be improved by a more thought-out approach to representing the visual data. The application currently tries to impose a single method for viewing data on the map, and this solution is hardly optimal for any viewing of the data. This situation can be greatly improved by embedding intelligence in the application that enables the application to determine which methods of viewing data are most appropriate. For example, when looking at data on a global scale, a visual interface that focuses on the more general location of the data is more useful than one that focuses on the exact location. The converse is true, as viewing happens more on a smaller scale, exact details become more important. A user interface that took these considerations into account would greatly enhance the utility of the program.

Finally, the ability to download data from GDTP servers on the Internet and upload that data into the actual GPS receiver would enhance the usefulness of this application. Currently, support is only in place for uploading data to GDTP servers. This support is primitive, as it does not attempt any translation between Garmin geographic types and general geographic types. A translation server would complement the application well, as it would be able to transform types such as roads and rivers into the simpler types supported by the receiver such as tracks and waypoints.

Chapter 4:

MycoMap

MycoMap is a web-based application that is intended to demonstrate the usefulness of an open GIS platform in the areas of ecology and taxonomy. Simply put, MycoMap is an online catalogue of the locations of various mycological species throughout the United States.

In contrast to a catalogue built by a single party, MycoMap is unique in that it uses existing trends and organizations in order to collect data. Mycology, the study of fungal species, is hobby and serious pastime for people around the world. Typical mycological activities include foraging for species in the wild, collecting samples of the species, and determining the taxonomy from the samples collected. MycoMap attempts to harness these activities by providing individual mycologists (amateur and professional) with an easy to use online tool that assists with cataloging, searching, and aggregating mycological data.

MycoMap is implemented as a web-based application. It is designed to be accessible across a range of platforms and browsers. The intended audiences of MycoMap are individual mycologists and mycological clubs. MycoMap stores and aggregates the data provided by the participants to provide a more comprehensive and complete picture of mycological populations than any single collection can. MycoMap provides incentives for mycologists to submit data by providing a number of features. The most important feature is a map-enabled application that allows mycologists to pinpoint the location of specimens without the use of specialized GPS equipment. MycoMap also provides quick and easy ways for mycologists to manage their collection data. Their data can be easily searched and organized based upon a variety of criteria, such as species, dates, and locations. Furthermore, MycoMap is designed to provide the foundation from which an online mycological community may grow. Users of the system can interact through discussion forums and by sharing collections. The combination of ease of use and community features is a strong incentive package to encourage use of the system.

Implementation Details

For the most part, MycoMap is a straightforward and typical community-centric Java Servlet application. Features such as the discussion forums and user logins are typical implementations that are fairly common across dynamic, community-driven websites. These will not be discussed in detail, but the actual source-code implementation details are available in the accompanying CD-ROM for interested readers.

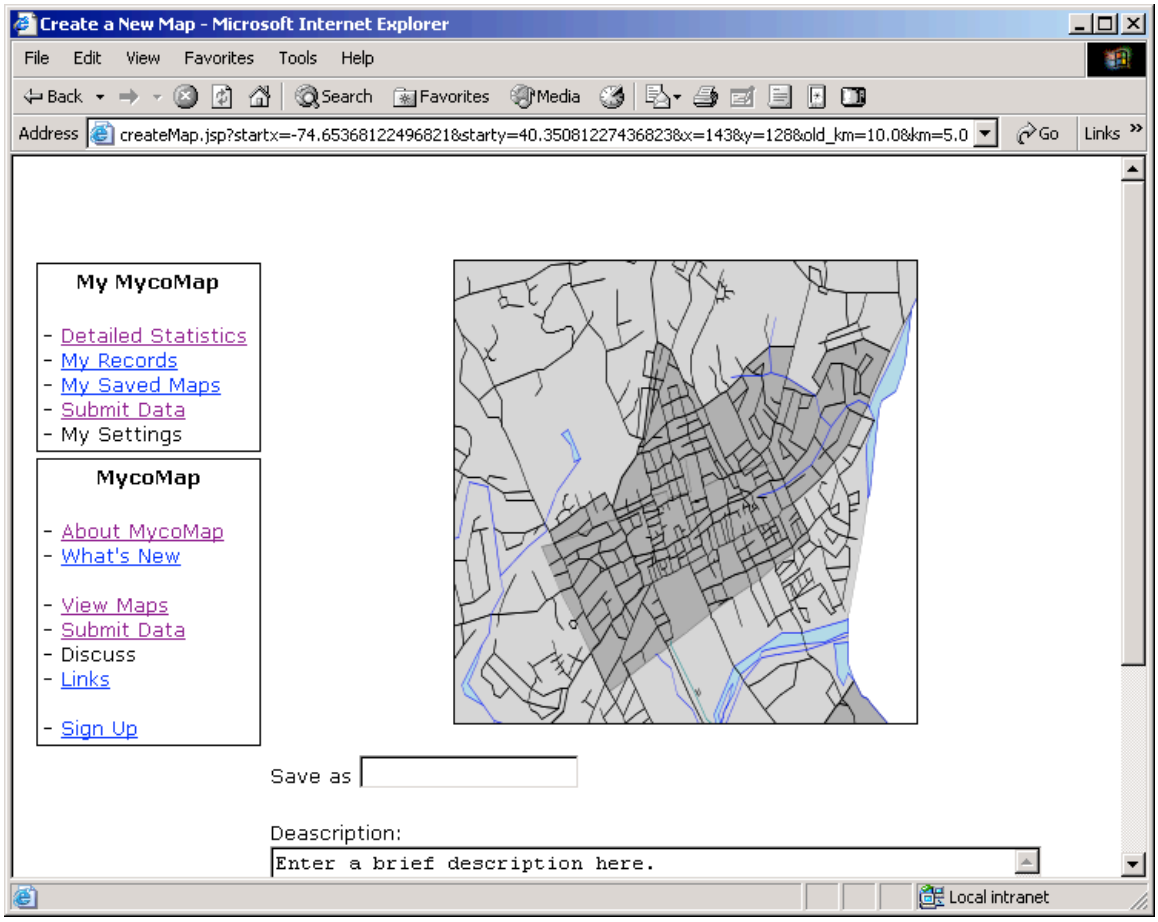

Figure 4.1: This is an illustration of the MycoMap application generating a map for later use. The map shown here was generated using the GDTP server described earlier and the MycoMap web-mapping software. This is a twenty-five square kilometer rendering of Princeton, NJ.

The interesting functionality within MycoMap with respect to this thesis is the mapping technology used to give users a convenient method for identifying the locations of species without knowing the latitude and longitude. The approach used by the mapping component of MycoMap is an intuitive interface where a user can select their general location and zoom into more specific areas until the desired resolution is found where the user can specify the location of the specimen.

The core technology that provides this functionality is a web-mapping component based upon a simple map-generation application. The map generator is a Java servlet that takes three parameters to generate a map: a center point longitude, center point latitude, and the range in kilometers that the map will span. A GDTP server provides the data used to generate the base map. This data consists of geographic features collected by the Census like roads, cities, and waterways.

The web-mapping component can be broken into three major functional components. The first component is responsible for parsing the HTTP request sent to the servlet container and extracting the location of the center point of the map and the kilometer range that the map will cover. Then, the component is responsible for determining the translation of the kilometer range into the correct latitude and longitude range that will be used to construct a bounded request to the GDTP server. With respect to the latitude, the ratio of degrees to kilometers is fixed, as the arc-length from one degree of latitude to the next is always the same, regardless of the longitude. With respect to the longitude, a more complex translation is necessary as the ratio of kilometers to degrees of longitude shrinks as the latitude moves away from the equator. Once the translations are complete, a bounded request encapsulating the area specified by the original query is constructed.

The second component of the web-mapping application is responsible for sending the GDTP request to a GDTP server and receiving the result of the request. The server's response is saved in a temporary file and the application applies an XSLT transformation that translates the GDTP response document into an SVG file.

Once the SVG file is generated, an image transcoder is applied to the file in order to translate the SVG file into a more accessible format. In this case, the SVG output is translated into PNG format. Once the transformation from SVG to PNG is complete, the PNG file is sent to the requesting client in order to fulfill the original HTTP request sent to the servlet.

This component provides the visualization of geographic features in a specific locale, but does not handle operations such as navigating over an area. Navigation features, such as zooming and panning, are provided by another servlet that uses the web-mapping component. A client using this servlet to generate and navigate through the maps provides the servlet with the location of the initial area to be examined and the range of the area. The navigation servlet displays an HTML page containing a hidden form and a map generated by the web-mapping component. The image is displayed as a clickable image map. When the map is clicked, embedded JavaScript code captures the click events and stores the location of the click within the hidden form. After the location is

stored, the form is submitted and a new page is generated. The new page has the location of the first map and the range of that map in addition to the coordinates of the imagemap click. Using this data, the application is able to translate the coordinates of the imagemap click into a new location relative to the original map. Furthermore, the range of the new map is decreased. The translated location and decreased range is provided to the web-mapping component, which generates a new map. The effect obtained is a zoom, where a user repeatedly clicks on a map and a new map is generated that zooms in on the clicked area. This process continues until the range of the query is small enough to determine the location of mycology specimens. Once this point is reached, the coordinates of the location are saved in the MycoMap database in an entry associated with the user to rebuild the map. The latitude, longitude, and final range are saved.

With the saved map data in the MycoMap database, users are able to build a list of maps of areas where their activity is focused. Using these maps, users are able to easily input their specimens. To input location data of specimens, users recall a previously saved map. An imagemap of the area is displayed. Users may click the map where the specimen was found and the application translates the click positions into actual physical locations.

This physical location data is used to generate a create feature request to a GDTP server that is hosting the location data for these species. A create feature request contains not only the location of the species, but also information such as the species name, the discoverer of the specimen, the time of discovery, and other metadata. This data is stored within the GDTP server in order that it can be easily mapped and is also available to other applications outside of MycoMap. Within MycoMap, generating maps of species on top of base maps is accomplished using the web-mapping component above and including mycological feature types within the feature list of feature types to be returned.

Non-Technology Considerations

The implementation of this application also has a number of non-technology issues that must be considered. The primary issue deals with how the data collected by MycoMap may be used and which parties should be allowed access to the data. Normally, this would not be an issue, but since more complete catalogues of locations and species types can enable more effective harvesting of specimens, great care must be taken in order to guarantee that the data is used responsibly. Furthermore, since some users may wish to restrict the data they provide to certain types of organizations, a layered classification of data is necessary, should the operators of MycoMap not claim ownership and complete control of the data being submitted within the terms of use. Some users may wish to only allow themselves access to their data, while others may be more liberal by allowing more parties to use their data. In this case, a rigorous application of the security framework hooks can be used to complement an existing data use policy. Since the mycology community is quite sensitive to these issues, an unambiguous data use policy must be provided in addition to the actual implementation.

Another issue is whether the precision of the location of the specimens should be artificially degraded in order to preserve the environment that hosts the species. Since some species of mushrooms are quite rare, it is natural that humans would seek to examine these species in the wild. However, since too many visitors to a certain location will have an adverse effect on the environment, it may be prudent to artificially degrade the precision of the location of the specimens to discourage others using the database as a means of finding and visiting specific specimens. This consideration should also be taken into account with the data-use policies.

Future Work

MycoMap offers a wide variety of areas that can benefit from future work. The most immediate is to generalize MycoMap so that it can be used to track species regardless of kingdom affiliation. In this vein, a more generalized MycoMap application can serve as common standard across similar applications. By making the application and interfaces to the application more general, the potential for a large multi-species information tool can be realized.

In the specific MycoMap application, future work in the areas of usability and "look and feel" would benefit the application immensely. By making the application more usable and more intuitive, it becomes more accessible to more people. In addition to usability, reengineering MycoMap as a language-neutral application with the appropriate translations would increase its utility worldwide. Mycologists worldwide would be able to contribute to the database, making the system truly comprehensive.

Furthermore, efforts to enable mobile devices the ability to access MycoMap is a large area for future work. Comprehensive functionality may be provided at the browser level, but limited functionality could be effectively implemented in handheld devices. For instance, tools for managing collections can be made available through a web interface, but functions such as submitting locations could be implemented on palm-top computers. These implementations would be dependent upon establishing an interface to the main application using technologies such as SOAP or .NET.

Chapter 5:

KeyHole

KeyHole is a prototype portable application that provides a real-time accounting of the location of a device running KeyHole software. KeyHole utilizes this open GIS architecture by using the network servers as a storage medium where the current and past locations are recorded. KeyHole uses a combination of a simple graphical client on the PocketPC platform[21] in conjunction with a servlet-based application that uploads the location data to a GDTP server and renders a vector map containing the location and surrounding features.

Implementation Overview

The servlet component of KeyHole is very similar to the graphics-generating component of MycoMap. The PocketPC component periodically uploads its location data by querying the servlet using a URL containing the current location embedded in the HTTP query string. The servlet component parses the query string for the location data, generates a custom GDTP query that creates a new location point on the GDTP server. After the successful creation of the feature, the servlet issues a response confirming the creation of the location point

The PocketPC component is an Embedded Visual Basic application that is connected to an NMEA 0813[22]-enabled GPS receiver. The PocketPC application periodically updates internal variables containing the current longitude and latitude information. During user-specified intervals, the application contacts the servlet component via a wireless network component, updating the location data periodically.

PocketPC Component Implementation

The PocketPC component[23] consists of a Compaq iPaq PocketPC device upgraded with a PCMCIA expansion pack containing a PC card with an RS-232 9-pin interface and the other card slot containing a cellular modem. The serial port is connected to a GPS receiver that can communicate via the NMEA 0813 protocol. The cellular modem is connected to a Nokia phone connected to AT&T's national cellular network. The GPS unit provides current location to the iPaq, and during set intervals, the iPaq communicates via the cellular network and local ISP to upload the coordinates to the KeyHole network component.

Figure 5.1: This is a screenshot of the KeyHole application running on a PocketPC 2002 emulator. The actual device used is a Compaq iPaq.

The software on the iPaq device is an Embedded Visual Basic application running on the PocketPC 2002 platform. The iPaq Visual Basic application uses standard ActiveX components to receive data via the serial port and send data over the cellular network. Internally, the application consists of three main components. The core component is the form containing the user interface and the internal variables that reflect the state of the machine. In addition to the form, two Timer objects are used to periodically handle certain operations. The first Timer object acts in intervals of one hundred milliseconds. Every interval it receives NMEA strings that have been buffered from the serial port and it parses the strings to update the local time and location variables. The second timer object operates in intervals of five minutes. Every interval it reads the local location variables and generates an HTTP query that informs the servlet component of the device's location.

The initial designs of this client did not have a server component, and the device would communicate with a GDTP server via normal GDTP requests to upload coordinate data to the server and obtain data about geographic features in the current locale. However, since the processing power required to transform a GDTP document to a graphical image was too high for the handheld devices, these types of operations were offloaded to an intermediate server that would generate the images then transmit the images to the client. Furthermore, since tools that are required for GDTP to image transformations, such as XSLT transformer and rendering libraries, are not currently available for the PocketPC platform, offloading the image generation to an intermediate server provided a rich suite of tools that sped the client development and provided an initial implementation of the translation server concept. Many of these features are not currently used by the Visual Basic component, but are implemented with the expectations that the PocketPC platform will soon be expanded and able to take advantage of the rich functionality offered.

Intermediate Server Component Implementation

The intermediate server component[24] assists the local PocketPC client with translating GDTP to a graphical format. Since the intermediate server is the software that communicates with the GDTP servers, efficient communication between the intermediate server and PocketPC client is achieved by allowing the PocketPC client to provide location and other data to the intermediate server via HTTP GET and POST requests. In this respect, the intermediate server is merely a web application that translates HTTP requests into GDTP queries and returns to the client images via HTTP. Used in this manner, the intermediate server is another class of application discussed below, called a translation server.

In practice, this component is implemented as a Java servlet application that receives an HTTP query string, parses the query string and builds a series of GDTP queries, then queries a real GDTP server. Upon the completion of the requests, the intermediate server can translate the results of the GDTP request into a graphical file, which can be returned to the PocketPC application for viewing. The implementation is straightforward and is almost identical to the web mapping components that are used in MycoMap. The

difference between this component and the MycoMap visualization component is that this component begins with sending GDTP upload requests to the server before sending bounded feature requests. In almost all other respects, the two components are identical.

Usage Details

Usage of this application is straightforward. In the main frame of the application, a number of configuration options are exposed that allow the user to dictate behavior of the application. The user can change the username associated with the tracker, the associated password, and the URL that the tracker uses to upload location information. Once this information is set, the user can enable and disable tracking by using the start and stop buttons.

For this application to function successfully, it assumes that the user has an existing Internet connection and that the GPS unit is configured as needed. The application has no facilities for doing either task.

Future Work

This software application can benefit from enhancing the functionality of the PocketPC application, creating derivatives of the application that run on embedded platforms, and eliminating need for the intermediate server.

Enhancing the functionality of the PocketPC client can be accomplished several ways. The most obvious enhancement would be to include more data that is uploaded to the server. The current client only uses a small set of the data provided by the NMEA protocol. Support for altitude, heading, speed, and even satellite status would give users tracking the location of the device more information about the device than just the location. This data can be used to paint a more complete picture about the device and its location. When coupled with other geographic data, information about the weather and surroundings of the device can be surmised. For instance, if the application were to upload the information about satellites and signal strength, other users accessing the data could reasonably guess the weather conditions around the device. If the signal strength was weak, despite being in an open area, observers could reasonably surmise that the device was under overcast or rainy conditions. If the signal strength was strong, despite being in the midst of a dense city, observers could reasonably guess that the device was under clear conditions. This idea of a third type of data being obtained from two unrelated types of data is potentially very powerful, and should be explored further along with the types of data that can be obtained this way and which types of data can produce the new types.

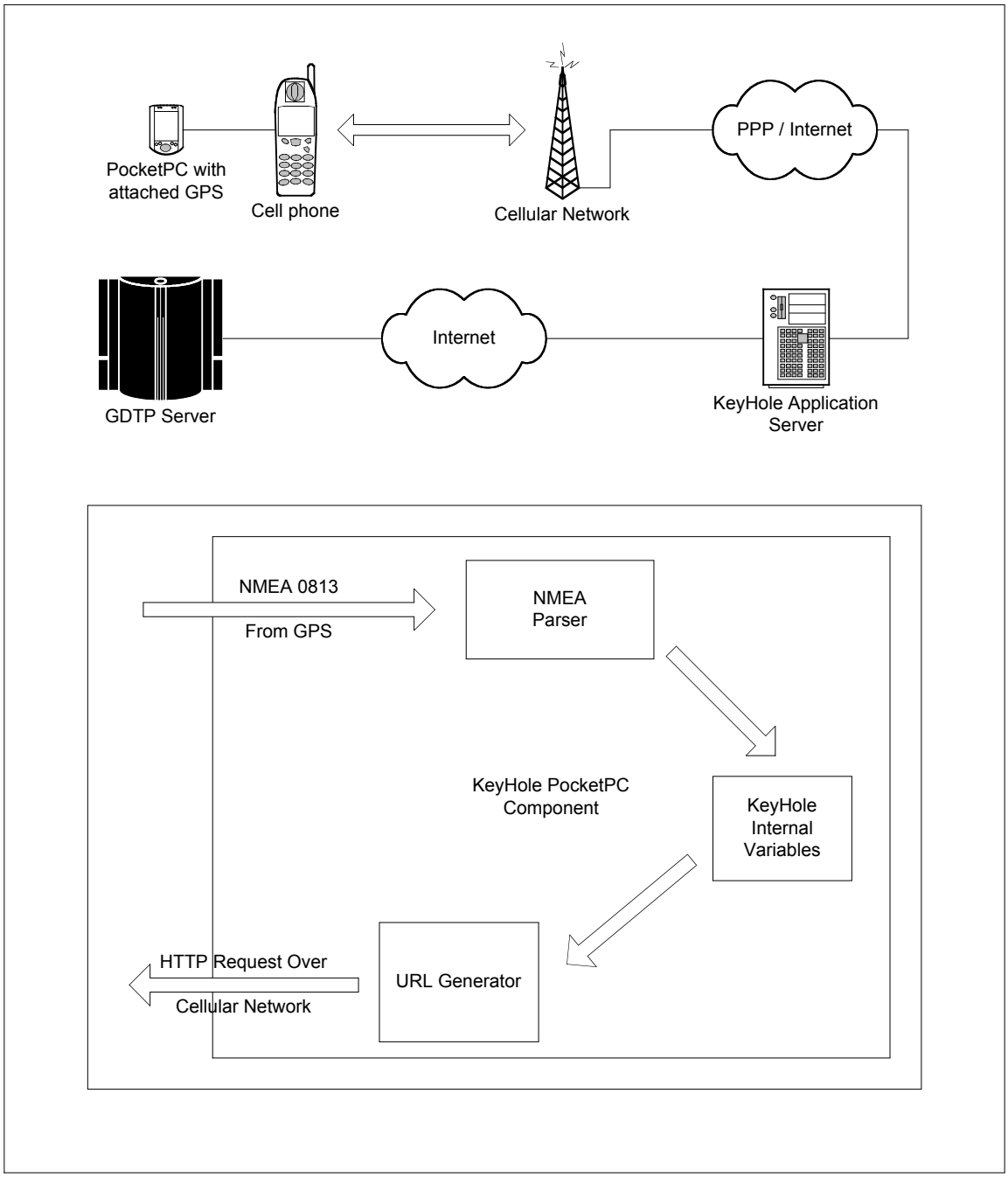

Figure 5.2: This diagram displays the architecture of the KeyHole system. The top diagram displays all of the components used to record locations. The bottom diagram details the internal architecture of the PocketPC component.

An additional enhancement to the functionality of this application would be the inclusion of security technology. The current implementation ignores the security and privacy considerations of the device and uses cleartext protocols that are easily intercepted. Features such as SSL tunneling for communications between the PocketPC application and the intermediate servers and the GDTP servers would go a long way to improving the security of this application. On the server side, a clear and precise statement of the security goals of the system must be articulated and implemented. Some systems will support open access to some trackers, and others will be more restricted. The appropriate analysis of the needs of each and how this is implemented on the application and server levels must be undertaken before deploying this technology in privacy or security-conscious environments.

Furthermore, the application would benefit from further work by taking advantage of image display technology that is available to Embedded Visual Basic applications. Originally, the application was intended to provide a graphic of its current location, but this functionality was not implemented in the client, as there were no standard libraries for displaying SVG, GIF, or even JPEG files within Embedded Visual Basic. It is only a matter of time before these components are made available. Once these are available, functionality in the intermediate server implemented with the expectation of being able to generate and visualize maps can finally be used.

Another area of further work involving this project is expanding the range of platforms that the PocketPC software runs on. In its current form, it is limited to a PocketPC platform with an Embedded Visual Basic interpreter. This software could be made reasonably portable across the entire range of Windows CE platforms (PocketPC, HandheldPC, AutoPC, consumer electronics) by rewriting the application in a lower level language such as C or C++ and compiling binaries for the different CPUs that support Windows CE. For example, to enable a system that can track the location of cars, this application can be rewritten for the AutoPC platform with the appropriate adjustments made for user interfaces and connectivity. A party with an interest in tracking cars like some of the rental car companies would be able to take the application and integrate it into existing AutoPC devices in the vehicles.

In the cases where an existing platform to run the software is not available, it would be feasible to take the rewritten application and design a custom hardware device around the application. This device would be relatively small, contain wireless networking capabilities, and include an embedded GPS receiver. By eliminating unnecessary features such as the user interface and backlit LCDs, it would not be difficult to create a programmable device that can be manufactured on a large scale and sold cheaply. Such devices would be used to add this location tracking capability to things like vehicles, briefcases, and personal accessories such as children's watches. By building upon an open and consistent platform for geographic data storage and retrieval, vendors of such devices stand to benefit by eliminating the costs of developing a dedicated proprietary architecture and providing an accessible interface for third-party developers.

Finally, the elimination of the intermediate server will eliminate the complexity of the system by reducing the necessary components from three to two. Further work in this area is heavily dependent upon further work elsewhere. In order for the portable devices to function as full-fledged GDTP clients, advances in XML technology on handheld platforms must be obtained. The missing technology that would eliminate the intermediate server is a fast and efficient XSLT parser for handheld platforms. If this were present, a handheld client could issue direct requests to the GDTP server then use XSLT to transform the GDTP responses into SVG, which is viewable on the client. Moore's Law and the current interest in XML technology on handhelds should make this component a reality in a reasonable amount of time.

Chapter 6:

Platform Extensions and Future Work

The applications presented in this thesis are several implementations of ideas that utilize this architecture for sharing geographic data. These applications are only a small sample of the potential application space. In the following pages, extensions to this architecture and applications that facilitate these extensions will be described.

GDTP Extensions

For the GarminGPSTool, KeyHole, and MycoMap applications, the current feature set of GDTP is adequate to implement the applications. However, there are many applications that would require additions to the current GDTP protocol in order to work on the platform. Extensions such as variable shapes for bounded requests, a transaction model, and support for GDTP proxys would allow new classes of applications to be built on the platform. Some of these applications will be described later in this section.

The current method of querying for features within a certain area is accomplished by issuing a *boundedRequest* query with the parameters of the opposite corners of the rectangle containing the features. This is adequate for many applications such as the ones described above. However, a number of applications would benefit if this restriction were lifted. One can imagine applications that use queries where the client specified a center point and a radius and features within the defined circle are returned. This type of extension is a subset of the larger number of topological and geographic operations that could be included as integral parts of an expanded GDTP protocol. Inclusion of these functions could also conceivably address some of the problems of features being within a bounded area, but not being returned as no defining points are within the bounds of the area. Inclusion of these types of functions in the server would benefit the overall efficiency of the platform, as the operations now must be done on the client side with the client requesting many extraneous features to satisfy the request. Including these features into the server can be accomplished by integrating or implementing a geographic/topological function library in the Java side of the server, or storing the geographic data within a spatially aware database that has support for such queries.

A notable feature that is missing in the current version of GDTP is a transaction model. Because an ACID-compliant transaction model is not present within the server, rollbacks during crash recovery are impossible. From the perspective of the client, the transaction support in the Web Feature Server specification would be well suited for GDTP. Implementation of this transaction model could consist of either tight integration with a backend database with strong transactional support and the client support to enable it, or

through the use of internal logging of operations to disk that can be used in the event of a crash, or the final solution could consist of a combination of the two.

Finally, the support of a proxying model that is similar to the one found in HTTP could result in the production and deployment of a new class of applications utilizing GDTP that act as both clients and servers. An immediate application of these features is the creation of caching proxy servers that could be deployed as part of the architecture in order to minimize the number of redundant queries sent to servers. Another application is a transformation server where the data format utilized by the client is different than the one provided by the server. By issuing a proxy-request through a transforming proxy, no changes in the way the client or server views the data is required. This is described in more detail below.

A number of other features could conceivably be included within the GDTP protocol. The ones described above do not exhaust the list of valuable features. It is expected that the protocol will be required to grow and evolve as the needs of spatial data users and their applications change.

Caching Proxy Servers

Currently, one of the problems that have been encountered in this project is the fact that geographic requests require an order of magnitude more computing power and memory than traditional and web requests. Every reasonable effort has been made to eliminate bottlenecks and enhance performance, but at the present, these servers are not nearly efficient enough to be placed on a public Internet and queried by multiple applications in real-time. In order to currently provide a robust and responsive enough architecture for the usage envisioned by this paper, significant investments in hardware or software are required. This disturbing trait runs counter to the intent of this project that barriers for allowing everyone to share geographic data should be as small as possible.

An alternative approach to utilizing expensive hardware and software is to build performance into the network itself by providing caching servers that replicate the data so that requests can be served efficiently through a network of caches rather than invoke an expensive query on the server. This approach obviously has a number of limitations, but it also shows a decent amount of potential.

One of the most fundamental limitations is that not all data can be cached as well as other data. For example, caching the location data that was involved in the KeyHole application would be a bad strategy because the data is updated much faster than a cache would be. Relying on a cache of this data would effectively raise the update interval from the original interval provided by the client to the update interval in the cache. Furthermore, while individual features can be cached easily and efficiently, areas of features are less easily cached. Caches can be coded with spatial intelligence with respect to caching bounded areas, but the cache must also be aware of the areas that are cached and which are not. The spatial intelligence can be improved to accommodate this fact, but

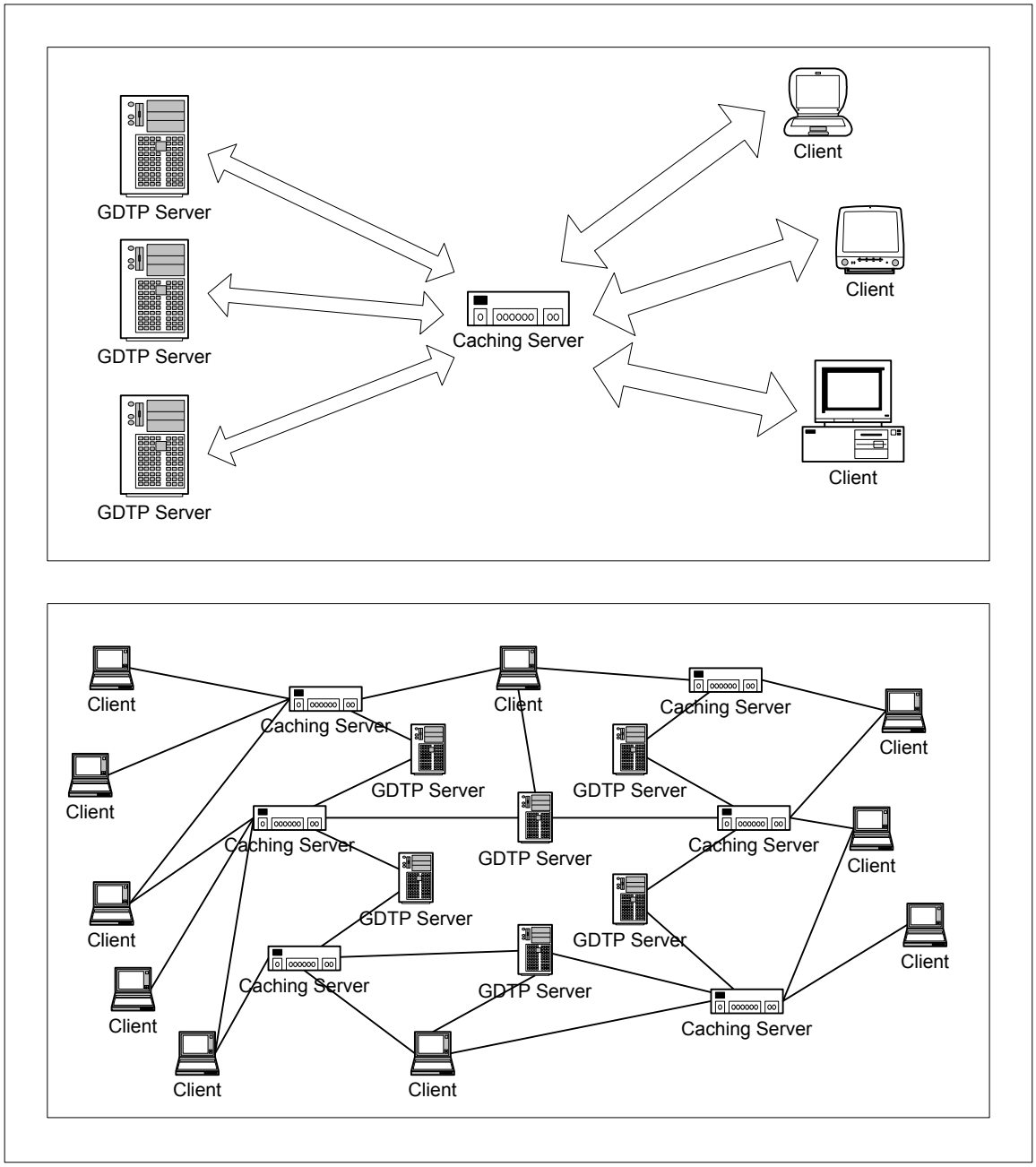

Figure 6.1: The top figure illustrates how a caching server can reduce the amount of traffic (represented by arrows) directed to the servers by caching recently accessed features. The second figure illustrates how an architecture incorporating caching servers serves to act as an intermediate layer to the core feature servers.

as more intelligence is added within the cache, resulting in a slower, more capable cache. If the area of features that is cached becomes significantly large, the question of whether it makes more sense for the cache to remain a cache or for the cache to become a full mirroring GDTP server.

The nature of geographic data also offers some potential for caches. One property of the data is that much of the data is effectively immutable and rarely changes. This type of data would benefit from caching as the cache has little work to do in order to provide current information. In this respect, the cache serves as an effective mirror of the original server, and periodically checks if the feature data is current. Furthermore, an intelligent cache can exploit that fact that certain geographic areas are of more interest to a larger user segment. A cache can exploit this by biasing its performance and providing preferential treatment to data that is requested often.

A number of possibilities exist with respect to applying caching actors in this architecture to enhance performance and reliability. A fair amount of future work can be devoted to determining the best behavior that the caches should implement, and how caches can efficiently cache data that is not necessarily single unrelated features, but whole areas of related and interdependent features.

Translation Servers

A translation server is the result the idea that certain types of coordinate systems are better suited for representing different types of data. For example, representing features that differ on a scale of meters is inefficient using the traditional degree notation because of the amount of precision required to represent the location is largely wasted, as the more significant digits are the same. In these cases, a more localized coordinate system would be more useful and efficient. This architecture can support this idea by providing uniform support to different coordinate systems through the use of translation servers.

A translation server is a simple piece of software that acts like a proxy because it tunnels the data from the original server to the original client, but it differs from a vanilla proxy as it has the ability to do coordinate transformations as the data is tunneled. For example, Garmin GPS receivers store location points using an integer system that is simpler than the traditional degree system, and arguably more efficient in some cases. It is not unimaginable that some servers would support this coordinate system instead of the traditional one. People using software designed to operate with the traditional coordinate system can use the data contained in the alternative system. In order to facilitate interoperability, a client could conceivably route its queries through a translation system that requested the data on behalf of the client and translated the coordinates before returning the data to the client. Such a piece of software would be simple to implement, and could be reasonably efficient. While the alternative to this approach is to encode the transformations within the servers themselves and allow the clients the ability to specify

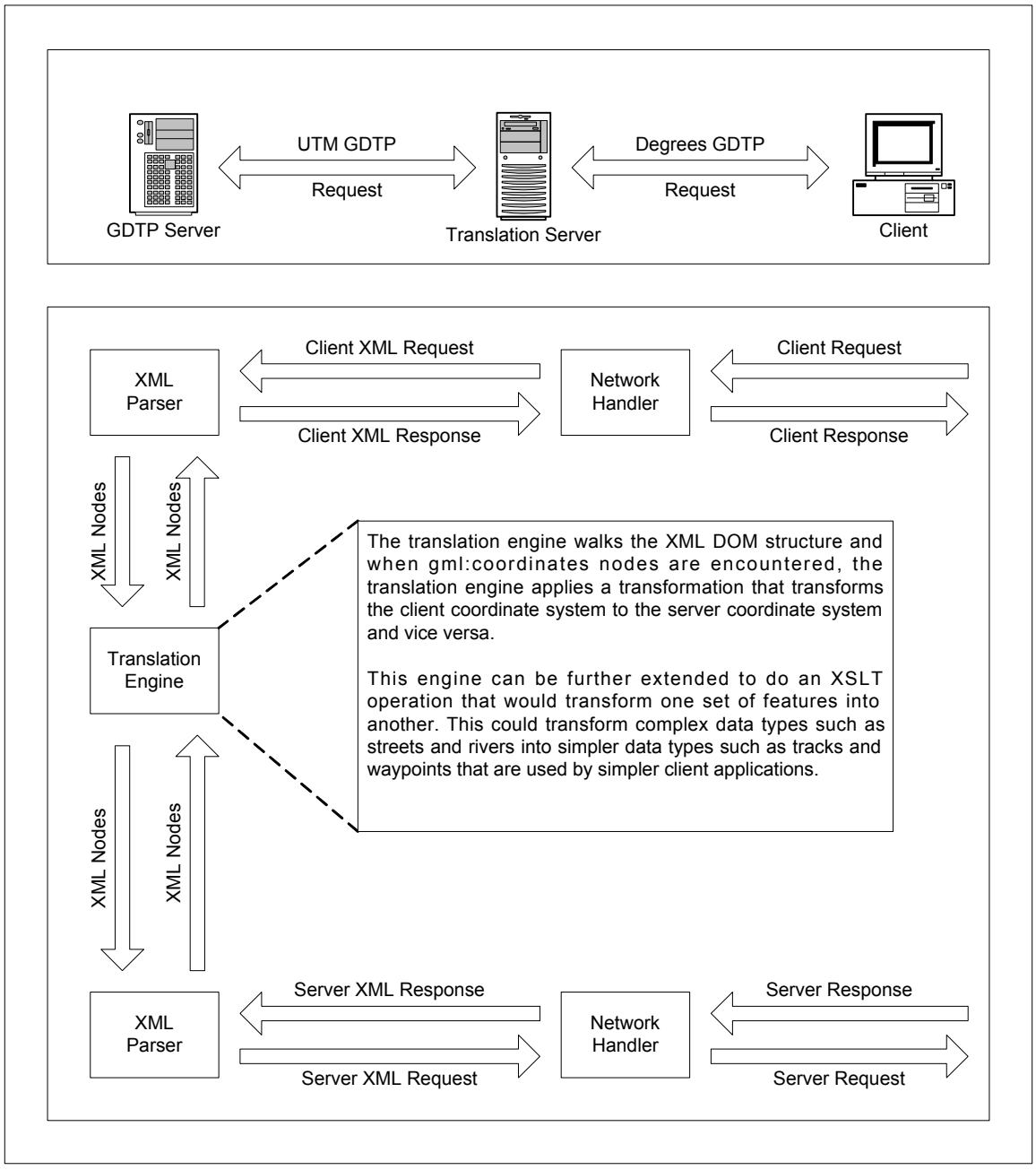

Figure 6.2: A translation server can translate between one coordinate system into another. Furthermore, translation servers can be implemented to translate from one GML data type to another by utilizing XSLT technology.

the coordinate encoding desired, the approach using translation servers is more flexible as it allows new coordinate systems to be deployed and used without requiring server software to be updated and existing data collections to be manipulated. From a performance standpoint, it may be beneficial to "outsource" the coordinate translations as servers become more overworked and in-server translations would only slow the response from servers to clients not requiring translated data. Since specialized coordinate systems are most likely to be used by only a small subset of applications, including translation capabilities in other software actors would limit the complexity of the servers without disallowing the use of these alternative systems.

Aggregation Servers

It is not hard to imagine that as more people start using GDTP software to share geographic data that the data landscape becomes richer and more varied. However, one problem is that as new servers are put online with new data, the task of knowing which data is available and contained on which server becomes an issue facing server operators and client application developers alike. The situation would be much like the World Wide Web without search engines – users would only be knowledgeable of a small set of all the servers available. One solution to this problem is aggregation servers.

Aggregation servers would conceptually function much like sites like Yahoo! that aggregate web content in order to provide visitors the ability find the respective site hosting the information. Aggregation servers would aggregate the content and provide a virtual collection of features that is actually resident on other servers. This could be accomplished transparently where the server acts as a semi-transparent proxy and relays the identity of the server to the client, or the proxy could be opaque and provide the data to the client as if it were the original feature owner.

An ideal application of this concept would be when the government shared road data for public use. Since maintenance and construction of roads is primarily a state responsibility, the state is the party best equipped to keep and maintain a database of roads. However, since there is an interest in a comprehensive national collection of roads for purposes such as making national maps, the federal government could represent the separate collections of roads as a single large collection through an aggregation server. One can also imagine communities that have complementary collections of data forming communal aggregation servers in order to advertise their data and facilitate access to the data.

Chapter 7:

Conclusion

The system described above has accomplished some of the goals outlined in the introduction. It is a platform-independent architecture for sharing geographic data across servers and applications. A number of client applications have been implemented that illustrate the features of the platform and show how different applications can use the platform in different ways. However, due to time pressures, the platform has not been developed to the point of being a completely viable application for general GIS systems. Work on solidifying the protocols and underlying components is necessary before any widespread adoption can take place. Furthermore, this project offers a wide variety of future work that can be done upon the server and client applications.

Some goals realized by this project that were not explicit from the beginning involved using new technologies such as XML and SVG as fundamental components of a complex software system. In this respect, this project has illustrated how these technologies can be effectively used in ways unintended by their creators. A good example of this is using the XML DOM tools as the internal data structure of the Garmin application. This type of activity shows how technologies can be used outside intended bounds to achieve new effects. In some respects, this concept is the central theme to this project and its products. By creating a general technology that is not artificially constrained by how the creator feels it must be used, creative programmers can achieve ends much wider than intended.

Finally, this project should be viewed as an initial attempt to tackle parts of the problems of creating and deploying a global unified geographic data space. This project tackled some of the implementation problems, but some of the larger problems in achieving this goal were not addressed. Specifically, determining how to format data across applications and data types will be particularly challenging. Furthermore, the task of creating compatibility across servers and applications will be challenging from both a technical and political standpoint. The technical problems will be less difficult than the political ones as standards emerge that encourage compatibility, but the political problems of convincing vendors to reverse the trend towards more focused and proprietary architectures will be potentially difficult. Hopefully by releasing open tools available under open source licenses, geographic communities of users will emerge and serve as a benevolent force towards establishing a unified geographic data space.

Appendix I:

Acknowledgements

The author would like to thank the following people and groups that were instrumental to this project's research. Without the help of the parties below, this thesis would not have proceeded as smoothly as it did.

Professor Brian Kernighan – Advisor

Professor Andrea LaPaugh – Second Thesis Reader

The Apache Foundation: Batik, Xalan, and Xerces Groups – Provided Major Technology Components

Rob Hranac, VFNY – OpenGIS Contact

OpenGIS Consortium – Provided GML and WFS Specifications

Five Blues Lake National Park – Provided Testing Grounds for Early GarminGPSTool Releases

Jaguar Creek – Provided Electricity and a Quiet Environment to Program in Belize

Princeton Round Table Fund – Provided Funds to Purchase Equipment

Appendix II:

Guide to Bundled Software

The thesis preceding this appendix is only a small part of the total scope of this project. The majority of the details with respect to implementation are available within the source code available on the accompanying CD-ROM. Since changes in the software and hardware platforms contribute to a situation where digital content often has a very limited lifetime, every effort has been made to include the software used during the course of this project and files that are encoded in open formats that are expected to best weather changes in a technical environment.

Third party applications released under open source licenses have been included on the CD-ROM under their respective licenses. Other applications released under alternative licenses, but freely available on the Internet at the time of writing have been included on the CD-ROM under the fair-use provisions of United States copyright law. Use of this software may be restricted and users are encouraged check the status of the licenses before using the software. These software packages are included in the 'Third Party Applications' directory so that future parties can compile and run the programs developed as part of this thesis.

The source code to project components such as the GDTP server has also been included. Instructions for compiling and running the respective software packages are included in the README files in each directory. Please note that these software releases are effectively development snapshots and that while the code will compile and run, it has not been thoroughly audited and bugs and development artifacts are present in the code. For more polished and up-to-date releases, check http://aetherial.net for new code.

.

Appendix III:

Bibliographic Notes and Endnotes

This document makes reference to a variety of terms, technologies, and implementations. The notes below explain some of the references and provide URLs for further research.

---

[1] 'GIS systems' is technically a redundant term, but GIS is used as a term to describe geographic technology, so the usage 'GIS systems' occurs often in GIS discussions.

[2] Open GIS systems in this respect are distinct from the OpenGIS Consortium. An open GIS system uses open and documented protocols to communicate between clients and servers. For more information about the OpenGIS Consortium, visit http://www.opengis.org/.

[3] When possible, the most recent version of the Java programming language was used. All Java code produced during the course of this thesis used the Java 1.3/1.4 programming tools. More details about Java2 can be found at http://java.sun.com/j2se/.

[4] SVG is a W3C standard for vector graphics. More information is available from Adobe's SVG site at http://www.adobe.com/svg.

[5] Batik is a product of the Apache Foundation's XML efforts. It provides native Java support for SVG-based display widgets in addition to transcoder tools that convert SVG to other graphic formats such as PNG and JPEG. Batik downloads and documentation can be found at http://xml.apache.org/batik/index.html.

[6] The GML 2.0 specification is online at http://opengis.net/gml/01-029/GML2.html.

[7] MySQL is an open source relational database system. More information about MySQL is available online at http://www.mysql.org.

[8] InnoDB is an alternative database backend that can be exchanged with MySQL's default MyISAM backend. InnoDB is much faster than MyISAM and provides transactional and performance features that are lacking in the standard MySQL installation. Information about this backend can be found at http://www.innodb.com.

[9] The source code of the development server has been included with the accompanying CD-ROMs. It should be noted that the source code provided is a development snapshot of the server used in this project. Major functionality is implemented, but the source code

has not been polished or packaged in a manner to facilitate easy deployment of this software. A more polished and tested release will be made available at http://aetherial.net.

[10] The bulk of the web interface is implemented in the *webAdmin.java* and *htmlPages.java* source files. At the time of writing, the web interface is still under construction and may contain bugs and other issues.

[11] This functionality is primarily implemented in the *geoserverXMLHandler.java* and *geoserverRequest.java* source files.

[12] Xerces is a programming toolkit for Java and C++ provided by the Apache Foundation. It is used extensively throughout the software in this project. More information is available online at http://xml.apache.org/xerces2-j/index.html.

[13] TIGER/Line is the system used by the United States Census to obtain, store, and disseminate geographic data. More information and data files are available online at http://www.census.gov/geo/www/tiger/index.html.

[14] XSLT is an XML technology for transforming XML documents of one type into another. This technology was used extensively to transform GDTP and GML documents into SVG. Apache's Xalan XSLT transformer was used throughout the project. More information about Xalan is online at http://xml.apache.org/xalan-j/index.html.

[15] Web Feature Server is an application protocol that is being developed by the OpenGIS Consortium. Its features and function are almost identical to GDTP. At the time of writing, the WFS specification is still in the design and testing phase and no public documents have been released.

[16] Secure Socket Layer (SSL) technology is security technology that allows for encrypted tunnels across public networks. More details about SSL is available online at Netscape Communication's website at http://www.netscape.com/security/techbriefs/ssl.html.

[17] Garmin manufactures a variety of GPS receivers. More details about Garmin GPS receivers is available online at http://www.garmin.com/.

[18] The proprietary protocol specification used by Garmin GPS receivers is online at http://www.garmin.com/support/pdf/iop_spec.pdf.

[19] The source code to the GarminGPSTool has been included on the accompanying CD-ROM. For more recent versions of this application, check http://aetherial.net.

[20] Information about the JavaComm API is provided by Sun Microsystems at http://java.sun.com/products/javacomm/.

[21] The PocketPC platform is one of Microsoft's embedded platforms that uses Windows CE technology. The primary development languages for the platform are Embedded Visual Basic and Embedded Visual C++. Information about the PocketPC platform is available at http://www.microsoft.com/mobile/pocketpc/default.asp.

[22] NMEA 0813 is a serial communications protocol implemented by many GPS manufacturers for use in environments such as ships and airplanes. NMEA 0813 devices provide a steady stream of data over the serial port that is translated into information such as current location, satellite status, and bearing.

[23] The Embedded Visual Basic project files for this application have been included on the accompanying CD-ROM along with the servlet component. Routines for reading and parsing NMEA 0813 strings were adapted from the GPSBoy application by Joshua Trupin. Project files for GPSBoy is included in the third party software section of the CD. For more information about GPSBoy, visit http://msdn.microsoft.com/msdnmag/issues/01/01/GPS/GPS.asp.

[24] The source files for the servlet component are available on the accompanying CD-ROM.