Distributed Data Collection for CyberTracker-Server

Anthea Peacock
apeacock@cs.uct.ac.za

CSC4016W

2006

Supervisor:

Professor Edwin Blake

Abstract

CyberTracker is a system that allows users working in the field to record data that they observed. This is done via an application that runs on a mobile device. Once the user leaves the field he or she has to go to a PC that runs the CyberTracker desktop application in order to allow for the collected data to be stored permanently. The CyberTracker desktop application allows for users to analyse data that was collected by querying and mapping the data.

The main aim of this project was to allow users to send the data collected on mobile device running the CyberTracker application to a remote database via GPRS. Enabling users to do this would eliminate the need for them to return to a PC containing the CyberTracker desktop application, thus increasing productivity and saving time. Querying the database is also an aim of this project as allowing users to have access to the information in the database may be useful while in the field. The specific task of this project was to develop a means for many client requests and sending of data to be handled by a server.

An iterative approach was chosen to develop the system. This means that analysis, design, implementation and testing were done throughout the project lifecycle. The analysis and design provided information about the required functionality and the operation of the system. This aided in the implementation and testing of the system.

A version of the system was developed in each iteration. These versions were tested and improved on before the start of the next iteration. This allowed the system to develop into a final product that was well tested.

The final testing of the system proved that the technologies chosen to implement the system were suited to the system. The technologies included a listener/server approach, the use of EJBs and XML.

The main aims of the project were met. These are allowing a server to handle many client requests and allowing a mobile client to update a remote database. The aim of allowing for the user to query the remote database in the field was not met. This was due to a number of project risks that became a reality during the project lifecycle. This may be a useful addition to the system if future work is done on the system.

Distributed Data Collection for CyberTracker - Server

1. Introduction

1.1 The existing CyberTracker system

The existing CyberTracker system consists of a client application that runs on a mobile device and a desktop application that runs on a computer. CyberTracker is used by people working outdoors making observations which need to be recorded. The type of observations collected can be changed using the desktop application to create a sequence that will allow any type of data to be collected. The CyberTracker application captures both observations, also known as sightings, and GPS location data, also known as waypoints.

Once the data has been collected in the field by using the mobile device it needs to be taken back to the computer containing the CyberTracker desktop application in order for the information to be transferred from the mobile device to the computer where it is stored in databases. The desktop application can also be used to query and map data.

1.2 Problem definition

The current CyberTracker system needs to be docked at a PC. This requires users to “return to base”. The extended CyberTracker system will allow users to send the data that was collected over GPRS to a remote database. This could help eliminate the need for users to have to go back to the computer and save them time and increase their productivity.

The specific task of this project is to allow a server to receive requests from the mobile device that will allow a remote database to be updated. The server must be able to handle many clients trying to update the database at one time. Users may also query the database to obtain information that will be useful for them in the field.

1.3 Aims and deliverables

The aim of this project is to provide a means for data to be sent via GPRS from a mobile device running the CyberTracker application to a server that will update a remote database which acts as a central storage location.

The specific deliverables of this project are:

- A means for many client requests to be handled.
- A means to update a remote database with the data that is sent by the client.

1.4 Structure of the report

Section two of this report is the background section. The background explores similar projects to this one and how communication between client and server took place.

Section three of this report contains the analysis of the project. The analysis section contains various software engineering artifacts used to describe how the system will operate and what the various components of the system are.

Section four of the report is the design section. The design section focuses on the overall system and serves an overview of the system components.

Section five of the report discusses the implementation of the project. Various implementation decisions made are discussed as well as the development of the implementation in each iteration.

Section six describes the testing that was done in order to evaluate the system.

Section seven contains the main findings that were discovered during the project.

Section nine is the conclusion section. The conclusion section discusses possible future work and also discusses the success of the project with respect to the project plan and whether the project aims were met.

2. Background

Many applications and projects exist that consists of mobile devices transmitting and accessing information to and from a database. Different client-server paradigms have been implemented to suit the requirements of these different projects. Among the client-server approaches used are the two-tier model, three-tier model and the extended client-server model.

In this section the client-server architectures of projects such as those mentioned above will be explored.

2.1 Projects using two-tier implementations

The two-tier client-server approach involves direct communication from the client and the server. Either the client or the server contains the application logic [2].

There are various mobile applications that utilize the two-tier model. These include C-notes [6], an application to access pharmaceutical information [1], 3D City Info [7], and a system to help surgeons to make decisions about organ transplants [3]. These systems basically communicate directly with a database via a database connector or via a scripting language when HTML is used.

Three systems that use database connectors are the C-notes system [6] and the 3D City Info system [7] and a system to help surgeons to make decisions about organ transplant [3].

C-notes aims to allow students to share ideas and take and highlight notes [6]. . Student's work (C-notes) can be sent from a handheld device to a C-notes server. All C-notes are saved as XML. A Java application generates a SQL query. Java Database Connectivity (JDBC) is used to perform the query resulting in the appropriate updating of the C-notes database.

3D City Info is a system that will allow mobile users to query a database to find the location of any place in a city [7]. Information is displayed by means of a 3D map. The system has been implemented on a laptop with aspirations to upgrade to the use of mobile phones using 3G technologies. Communication with the City Info relational database takes place through JDBC via Java applets.

The system that helps surgeons make decisions about organ transplants allows surgeons to use handheld devices to access information about organ receivers [3]. This information is used to aid the decision of whether the transplant should happen and who should receive the organ. This is useful because on average half of the time the surgeon is not in the hospital at the time that an organ becomes available. If the surgeon is not in the hospital he or she will have to come into the hospital and look at relevant information wasting time. In this system the handheld devices communicate with the database containing organ receiver information via an open database connectivity (ODBC) interface.

A system that uses a scripting language one that was developed to allow doctors to access a pharmaceutical database by using a cellular phone [1]. The database containing the pharmaceutical

information is a MySQL database. SQL is sent to the database via PHP, which is embedded in HTML applications on the cellular phone.

Using the two-tier model poses a restriction in the sense that a two-tier system is not very scalable, as each client needs to have a session with the database. The database will not be able to manage many concurrent users accessing or updating it. Therefore mobile applications that use the two-tier model must not have many clients accessing the database simultaneously.

An advantage of using the two-tier model is the fact that it is simple to implement and it may be sufficient in environments where users do not necessarily access the database simultaneously.

Using the two-tier model for some of the above systems may be inadequate. In the cases of [1,6,7] there may be many users wishing to use the system simultaneously and using a two-tier implementation may be inadequate.

In the case of [3], where surgeons need to access information about organ receivers, using a two-tier system would be adequate as organs generally do not become available frequently.

2.2 Projects using three-tier implementations

The three-tier approach involves another layer between the client and the server. The application logic can now reside in this layer, although some application logic can also reside on the client or server [4].

Two systems that use a three-tier approach are the ENVIT system [9] and FieldNote [8].

ENVIT is a system that allows for the collection of environmental and geolocation data. The system encompasses a concept called ‘field data streaming’ [9]. This means that data is both sent and received to and from a remote server.

The data collected using the ENVIT system is configurable by the user by customizing a master database where all data gets sent. The master database resides on a field server on a laptop computer. Data can include various environmental sensor data, measurements and GPS co-ordinates. A copy of the master database is saved on the handheld device. When data is sent from the handheld device to the master database only data that has updated or changed is sent. Both the local and master databases are SQL Server databases. In order to send data from the local database to the master database a SQL Server CE Mobile Client Agent on the handheld device interacts with a SQL Server CE Server Agent on the server side. The SQL Server CE Mobile Client Agent and SQL Server CE Server Agent acts as a middle tier operating between the handheld device and the server.

FieldNote is a system that incorporates handheld computers in collecting and re-using data in the field for use in the environmental and archaeological sciences [8]. With FieldNote users can send data collected on handheld devices including FieldNotes (notes written by fieldworkers in the field), GPS co-ordinates and attribute data (such as counts) to a database. The database is accessed through a Web server (Apache) and Java servlets. The Web server acts as the middle tier in this case.

Unlike two-tier systems three-tier systems allow for scalability. Thus many users can use the system simultaneously. In the cases of FieldNote and ENVIT their may be teams of field workers accessing the database concurrently, so the use of a three-tier system is appropriate.

2.3 Projects that use the extended client-server model

Mobile computing differs from traditional wired computing. The first way in which they differ is the fact that mobile users are not stationary. They can move around and may get disconnected when this happens [2]. Another way in which they differ is the fact that mobile users could have less bandwidth than the wired users. Issues of battery life also come into play as it could hinder connectivity [2]. Due to all these factors traditional approaches to client-server architectures may not be advisable. An approach called extended client-server model or flexible client-server model is one approach that aims to deal with these factors.

Extended client-server model aims to allow the mobile client to have some server capabilities. This is done so that the mobile client can operate when connectivity is lost. When a connection is available the mobile client will function as a normal client [2].

The extended client-server model is used in MET (Mobile Emergency Triage) [5]. MET is a clinical decision support system (CSSD). When new patients arrive in a hospital clinicians are required to decide what type of treatment patients need. They are often pressed for time and may have to go to a desktop computer containing a CSSD. This could waste time. Having the system available on a handheld computer could be convenient for clinicians, as they would not have to consult a CSSD residing on a desktop computer.

In the MET system some components are stored on the client side in case of a disconnection. These components are the triage support component (to assist in decision making) and the patient database. Otherwise the client communicates with the server that contains a temporal database. Usage of the temporal database allows for efficient communication between client and server where updates of the database only happens when completely necessary.

The MET system is critical as it is used in emergency cases in hospitals. The use of the extended client-server is appropriate as the clinician cannot afford to not have access to important information.

In systems where access to information is not critical it may not be feasible to use the extended client-server model. One reason for this would be the fact that server capabilities on the mobile phone may not possible or infeasible. An example of this is where the mobile client is accessing extremely large database tables. Replication of these large databases on a mobile device may not be possible as there are space limitations.

2.4 Conclusions

The three different approaches described above have its advantages and disadvantages. Choosing an adequate implementation depends on the system’s characteristics.

Two-tier may be feasible when few users are going to use the system. It is a simple approach that is easy to implement. So, if the system does not need to be scalable then using two-tier could be sufficient.

If the system is needs to be scalable because many users will use it simultaneously or the user-base is expected to grow, then the three-tier architecture could be used. Mobile clients do have the problem of weak connectivity so the three-tier system may fall short in this respect. If data access is not critical then using the three-tier system could still be feasible.

The extended client-server model deals with the weak connectivity issue. So, applications that require urgent access to information could use this model. The ability of the mobile client to act as a server may however be limited as mobile devices generally have limited resources.

Using the three-tier approach is suitable for the CyberTracker system as many users may access the CyberTracker server at once. CyberTracker currently has a database on the mobile client. This database saves all sightings when they get recorded. So, the extended client-server model is already used in the CyberTracker system.

2.5 References:

[1] Hansen, M. and Dorup, J. Wireless access to a pharmaceutical database: A demonstrator for data driven Wireless Application Protocol (WAP) applications in medical information processing, *Journal of Medical Internet Research*, 3, 1, (2001)

[2] Jing, J. , Helal, A.S. and Elmagarmid, A. "Client-Server Computing in Mobile Environments," *ACM Computing Surveys* ,31, 2 , (June 1999) ,117-157.

[3] Krause, A. , Hartl, D. , Theis, F. , Stangl, M. , Gerauer, K. E. and Mehlhorn. A. T. Mobile decision support for transplantation patient data. *International Journal of Medical Informatics*, (2004) 73, 5, 461-464.

[4] Lewandowski, S. M., "Frameworks for component-based client/server computing,"*ACM Computing Survey*, 30, 1 (March1998), 3-27.

[5] Michalowski W. , Slowinski R. , Wilk S. , Farion K. , Pike J. and Rubin S. Design and development of a mobile system for supporting emergency triage. *Methods of Information in Medicine 2005*, 44, 1 ,14-24.

[6] Milrad, M. , Perez, J. and U. Hoppe. C-Notes: Designing a Mobile and Wireless Application to Support Collaborative Knowledge Building. *Proceedings of IEEE International Workshop on Wireless and Mobile Technologies in Education*, (August 2002), 117–120.

[7] Rakkolainen, I. and Vainio, T. 2001. A 3D City Info for Mobile Users. *Computer & Graphics 25*, 4, (2001), 619-625.

[8] Ryan, N. , Pascoe, J. and Morse, D. Fieldnote: A handheld information system for the field. *In proceedings of 1st International Workshop on TeloGeoProcessing* (Lyon, France, May 1999), Claude Bernard University, 156-163.

[9] Vivoni, E.R. and Camilli, R. Real-time streaming of environmental field data, *Computers and Geosciences*, 29,4 (2003) 457-468.

3. Analysis

This section contains the analysis of the system. This analyisis was used in order to implement the system and obtain the functionality of the system. Specific software engineering artifacts were developed which were associated with the various requirements of the system. Various diagrams were used to illustrate the different components and approaches used in this project. These diagrams include a use-case diagram, sequence diagrams, a state diagram and a class diagram. This section also has a risk analysis. The risk analysis was found to be very important for this project as many risks were associated with this project.

3.1 System Requirements

From the problem statement three key requirements were identified. These requirements would serve as the basis for all other analysis to take place. The requirements are:

- The system must be able to accept sighting and waypoint data from a mobile device and update a remote database with this data.
- The system must be able to handle many simultaneous clients trying to send data simultaneously.
- The system must be able to send acknowledgements and errors messages to the mobile device to allow the mobile client to keep track of what data had been received and inserted into the database.

3.2 Use Case Diagram

From the above requirements three high-level use cases were identified. They are:

- Mobile client sends a request to send data.
- Mobile client sends a sighting.
- Mobile client sends a waypoint.

Figure 1 shows the high-level use case diagram which represents all interactions with the system.

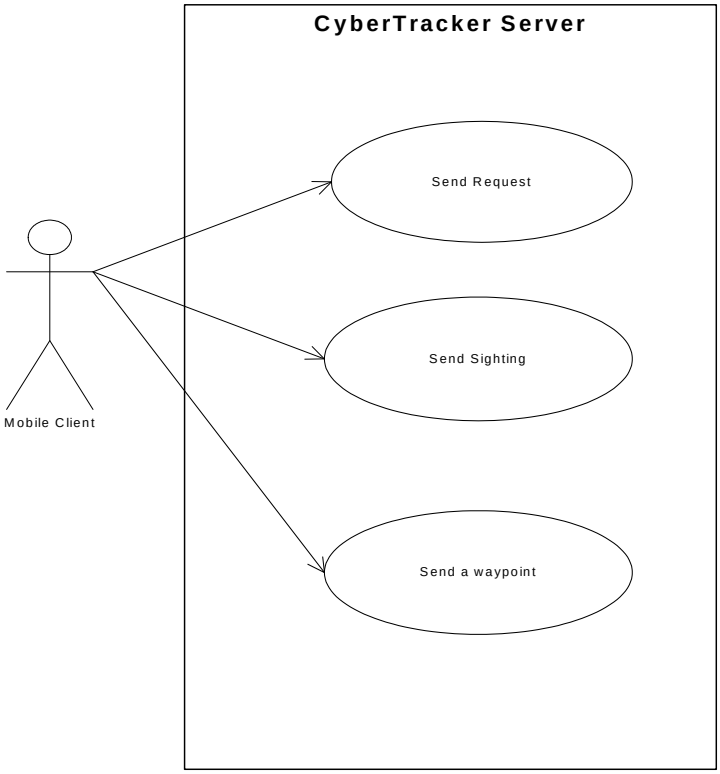

Figure 1: High-level use cases. These represent the main interactions with the system.

3.3 Protocol for sending of data from mobile client to server

3.3.1 RTS and CTS messages

A protocol for sending data from the mobile client to the server was decided on by using the use-case diagram above as a basis. It was decided that the mobile client would have to send a request to the server before actually sending sighting or waypoint data. The reason for this is because the server may be offline when the client tries to send a sighting or waypoint. The client would have to establish if the server could handle its request before proceeding to send data. In order to do this the server must send a message back to the client to let the client know that the server is ready to accept data. Once the client has received this message it can proceed to send the sighting or waypoint data.

The request message that the client sends to the server is called a RTS (request to send) message. The message that the server sends to the client to indicate that it is ready to receive data is called a CTS (clear to send) message.

3.3.2 The Listener component

In order to handle the requirement that states that the server should be able to handle many

simultaneous client requests a listener component has been introduced. The listener waits for client requests and once it receives request it spawns a server that will handle the request. In this way many clients can be making sending data at one time.

3.3.3 Steps of the protocol

1. The client sends a RTS to the listener.
2. The listener receives the client request.
3. Listener spawns a server.
4. Listener continues to wait for other clients.
5. Server sends a CTS to the client.
6. The client sends a sighting or a waypoint.
7. The server receives the sighting or waypoint.
8. The server sends an acknowledgement to the client.
9. The server exits.
10. The client receives the acknowledgement and exits.

The steps in this protocol are illustrated by Figure 2. Figure 2 is a sequence diagram which shows the interactions between the client, listener and server.

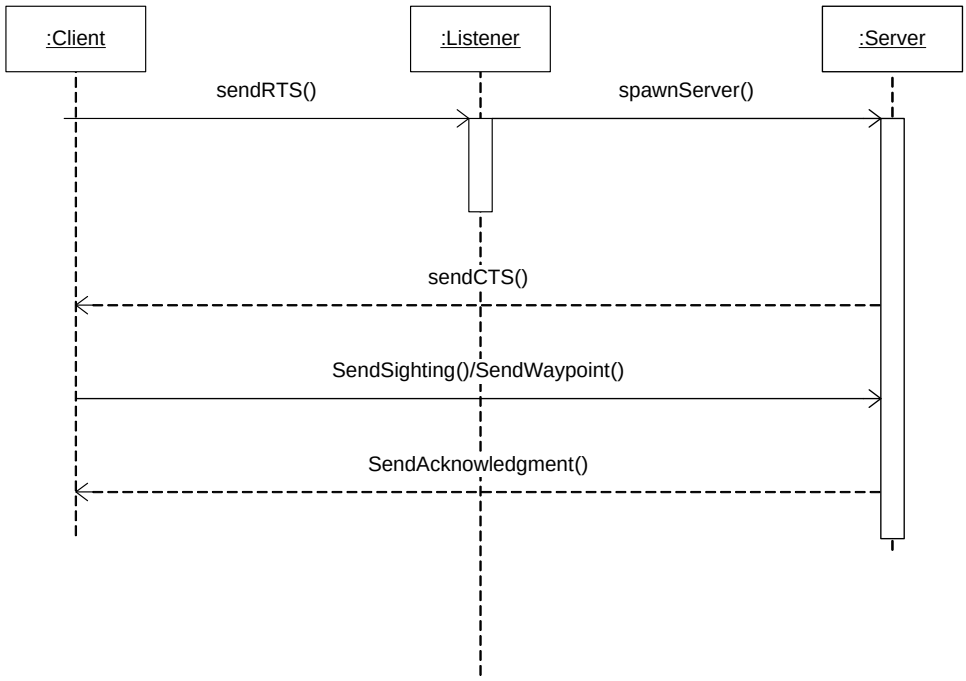

Figure 2: Sequence diagram representing the sending of sighting or waypoint data.

Figure three is a state diagram representing the state of the listener. The listener only has one state where it is waiting to receive a request. Once a request is received it spawns a server and goes back into the waiting to receive request state.

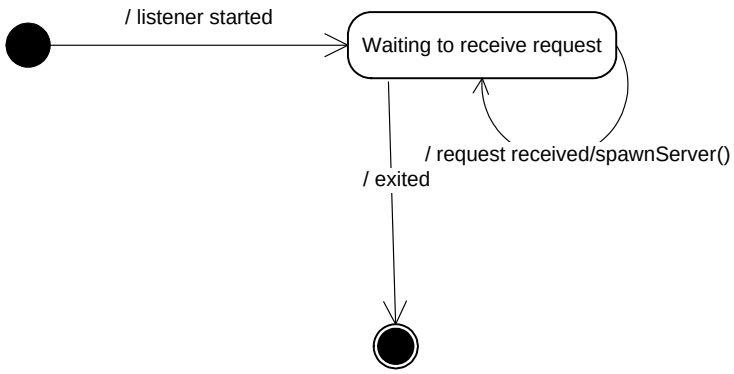

Figure 3: State diagram representing the state of the listener. The listener is able to service many client requests as it spawns a server to handle each client as requests come through.

3.4 Using EJB to handle simultaneous database updates

With the server being able to handle many clients, simultaneous updates of the database has to be considered. If many clients attempt to update the database at one time then transactions need to be implemented to make sure that one client does not try to update the same tuple in the database as another client.

By using container-managed Enterprise Java Beans (EJB) transactions can be handled by the container. This will ensure that every EJB method invoked will be treated as a transaction.

Three EJBs were proposed to handle the updating of the database. These include two entity beans: namely Sighting and Waypoint2. The Sighting Bean and Waypoint2 Bean correspond to the Sighting and Waypoint2 tables which are in the current CyberTracker databases. The Sighting table contains all sightings that have been docked using the CyberTracker desktop application. The Waypoint2 table contains all waypoints (GPS location data) that have been docked using the CyberTracker desktop application.

The other EJB proposed is a stateful session bean called CTSession that is used to invoke the above-mentioned entity beans.

With the decision to use EJB the project team had to come up with a way for the mobile client and the server to communicate in a language-independent manner. The reason for this is because the application on the mobile device is written in C++. No Java or C++ types could be sent as either the server or mobile client would not be able to decode what was being sent. It was then decided to send byte streams from the client to the server and vice versa. Once this was decided a way to

represent when one field of data stopped and another one started. It was clear that some kind of escape sequencing was needed. The team then decided that XML would be used as a means for allowing data fields to be packaged and to be discernible from others. It was decided that a class called Message would do all the creating and parsing of XML messages.

Figure 4 is a sequence diagram similar to figure 3 except that the CTSession Bean is included to show how the server will access and use the CTSession Bean. The CTSession Bean will access the Sighting Bean which will result in an update of the Sighting table in the database. The same process will be used when the client sends a waypoint except that instead of the Sighting table being updated, the Waypoint table is updated.

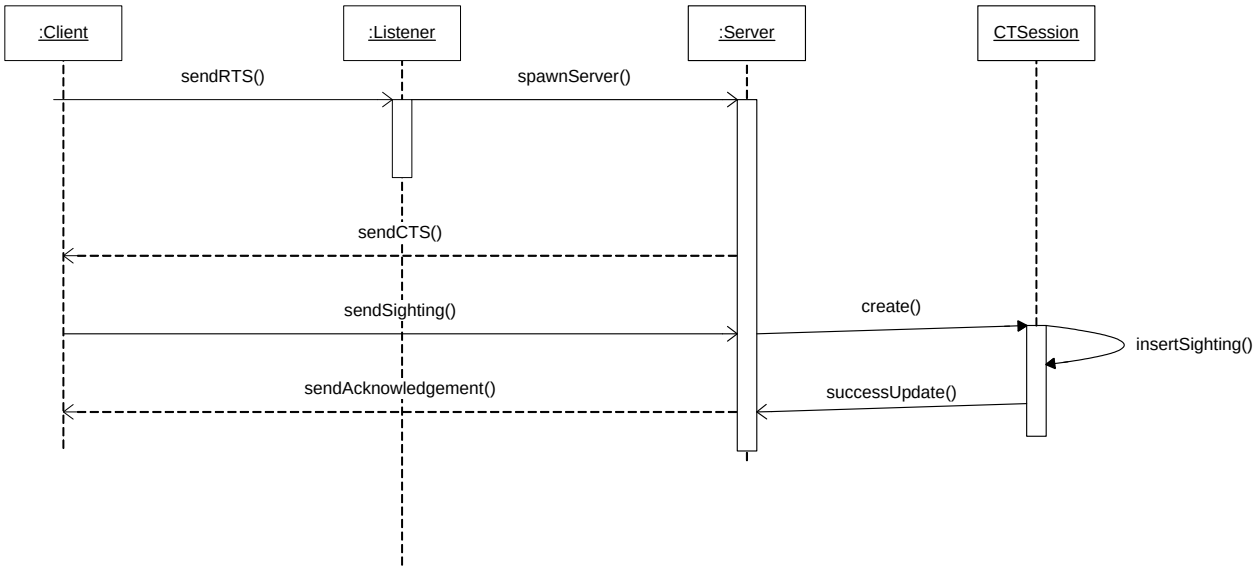

Figure 4: A sequence diagram similar to figure 2 representing the sending of a waypoint. This sequence diagram shows the interaction between the client, listener, server and CTSession Bean. The CTSession Bean will invoke the Waypoint2 Entity Bean which will allow for updating of the Waypoint2 table in the database.

3.5 Conceptual Class Diagram

From the above analysis six classes were identified. These are:

- Sighting
- Waypoint2
- CTSession
- Waypoint2
- Listener
- Server
- Message

The relationships between these classes are all association relationships. One Listener has a association relationship with one or many servers. The association relationship was named “spawns”. As mentioned above a listener spawns a server for each client request. One Server has an association relationship with one CTSession. The association relationship was named “uses” as a server looks up a CTSession Bean and invokes some of its methods. The same applies for Waypoint2. The association between Server and Message is also called uses as the Server class uses the Message class in order to manipulate strings into XML messages and parse XML strings.

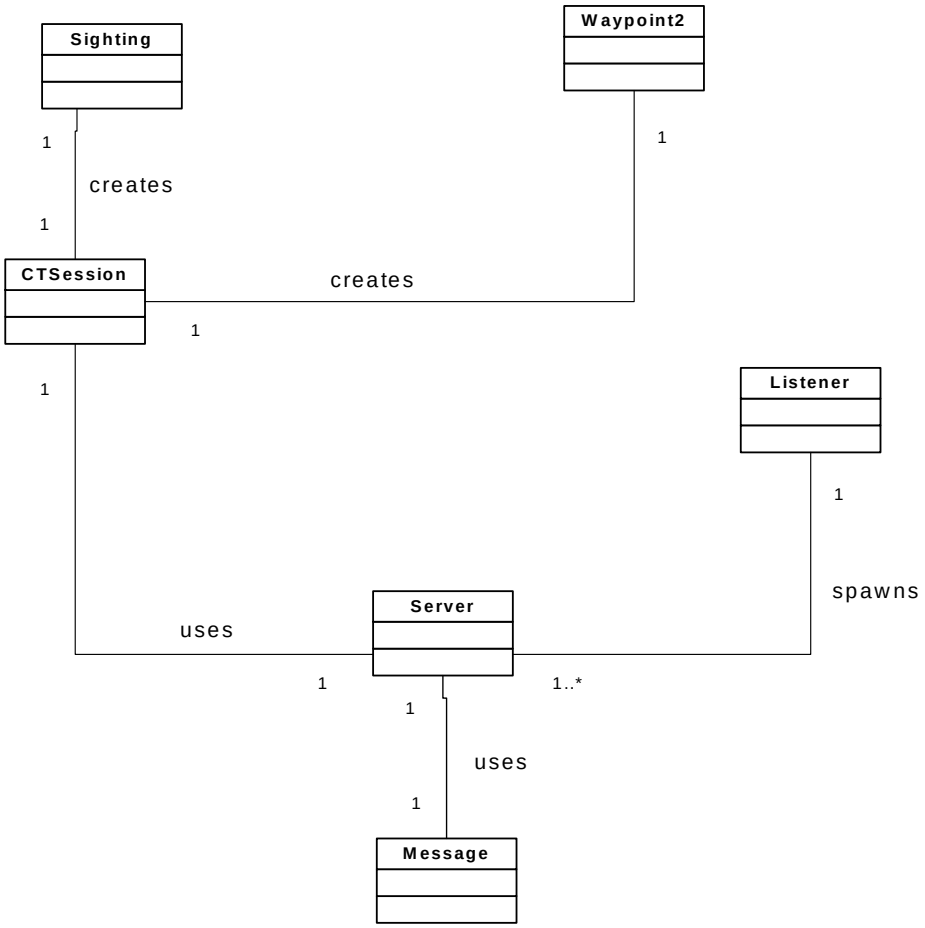

Figure 5: Conceptual Class Diagram. There are five classes. The relationships between these classes are named associations. One example of this is the Listener which spawns many Servers.

3.6 Risk Analysis

Several risks were identified in this project which needed a monitoring and mitigation plan. Table 1 is the risk probability matrix which was used to categorise the effect that the risk would have on the project. A is the most probable and severe risk and F is least probable and severe risk.

		Probability		
		Low	Medium	High
Impact	Catastrophic	C	B	A
	Critical	D	C	B
	Marginal	E	D	C
	Negligible	F	E	D

Table 1: The Risk Probability Matrix used to categorise the urgency of a risk. A is the most urgent risk and F is the least urgent risk.

Table 2 shows the various risks that are associated with the project. All risks have been categorised with B and C symbols. Most risks have a high probability of happening. This shows that mitigation and monitoring of the risks is vital to the success of the project.

The most severe risk of the project was whether the project team would be able to understand the existing CyberTracker code, architecture and database. Integrating the code written into the existing CyberTracker code was very important in this project as this project aims to extend the current CyberTracker system. The project team needed to understand the existing code before they could integrate with it. So, understanding of the existing code became the main risk which had to be managed in order to ensure the success of the project.

Risk	Impact	Probability	Category
Disk space provided for the CyberTracker code cannot be used for compiling or editing data.	Critical	High	B
No access to the existing CyberTracker code.	Critical	High	B
Second supervisor ill.	Marginal	High	C
Lack of adequate semi-literate users (preferably animal trackers)	Critical	High	B
Difficulty in integrating the new code with the existing CyberTracker code.	Critical	High	B
Expense of using GPRS to test.	Marginal	High	C
C++ Emulator has no access to the internet.	Marginal	High	C
One member falling ill or dropping out of the project.	Critical	Medium	C
Bad connection due to the use of GPRS.	Marginal	High	C
Difficulty of team members in trying to understand the current CyberTracker code, architecture and database.	Critical	High	B

Table 2: Categorisation of the project risks using the Risk Probability Matrix.

Table 3 shows the mitigation and monitoring of risks. As mentioned above a risk management plan was vital in a project that faced many risks. Most risks were successfully managed although some risks did present a problem.

The risk of understanding the existing CyberTracker system became a serious and probable risk. The existing code had no supporting documentation and few comments. The team had to spend many hours trying to figure out the code. The team decided to keep in contact with the developer of the CyberTracker software in order to get some help and advice. Contact was made with the developer but response was slow as email was being used as a primary means of communication. This caused the progress of the project as the team had no documentation to fall back on if they could not understand the code.

One example of this was encountered when trying to figure out how the CyberTracker database worked. CyberTracker uses Microsoft Access for its databases. The database consists of many tables, namely: Media, Settings, Sighting and Waypoint2. It was obvious that the tables that are used to store the actual sighting and waypoint data was the Sighting and Waypoint2 tables. It was however not obvious as to how the specific sighting data is stored. The sighting data was stored in a column called Data. The column was however a BLOB object. It was thus impossible to see what data the BLOB object actually stores. The existing code written to handle the database was written in Delphi and the point in the code where the sighting table was updated could not be found. The developer was contacted via email and after a few emails it was found that the sighting data contained a list of GUID and value pairs. The GUID referred to one element chosen on the CyberTracker application and the value was the number entered to indicate the quantity of the element.

Risk Condition	Consequence	Mitigation	Monitoring	Management	Update
Disk space provided for the CyberTracker code cannot be used for compiling or editing data.	The entire system, including the newly written code and the existing CyberTracker code cannot be built together.	The project group needs to build individual components without the integration with the exisiting CyberTracker code.	Check daily if the situation has changed by attempting to compile a small program and contacting the network administrator.	Consulted supervisor and told him about the problem. He offered to buy an external hard drive.	An external hard drive was bought and project team was able to build the entire system successfully.
No access to the existing CyberTracker code.	Inability to test whether the code written can be integrated with the existing CyberTracker system.	Proceed with the writing code and use the CyberTracker system to estimate where code written would fit in.	Check daily as to when the CyberTracker code will be be available.	Contact second supervisor and ask him to place the CyberTracker code on the system.	The existing CyberTracker code was made available to the project team. The second supervisor placed it on the external hard drive.
Second supervisor ill.	The project team is unable to get advice	Proceed without the advice of the	Ask first supervisor about second	Rely on first supervisor as a primary means	Second supervisor was ill for most of

	about the CyberTracker system.	second supervisor and try to figure out how CyberTracker works by consulting the CyberTracker website.	supervisor. Stay in contact with second supervisor via email.	of communication about CyberTracker and advice.	the project.
Lack of adequate semi-literate users (preferably animal trackers)	The member working on the usability of the system may not be able to test the system.	The team member needing to test usability can do research on the required test subjects.	Enquire about the availability of test subjects. If the test subjects are unavailable then focus on getting alternative test subjects.	Ask the owner of the CyberTracker system about the availability of users and get possible dates and contact numbers.	The member working on usability was unable to get semi-literate test subjects. Testing was done on other test subjects who might find the CyberTracker useful.
Difficulty in integrating the new code with the existing CyberTracker code.	The components written by team members will be separate from the CyberTracker system. This will mean that the components cannot be tested.	Each member must write modular code in order to make easy changes.	Members needing to integrate code must integrate at the end of each day.	Team member must make a concerted effort to integrate his or her code and ask for help if unable to do so. The developer of CyberTracker should be contacted if there are problems with integrating.	There were many problems at different times with trying to integrate code. The team was successful in integrating after many attempts.
Expense of using GPRS to test.	Unable to connect to the internet and thus unable to send data from client to database.	The WiFi network can be used instead of the GPRS network.	Monitor the amount of airtime used when sightings get sent.	Use WiFi to send sightings and use GPRS only for certain test cases.	The WiFI network was the primary means of sending and thus the cost of GPRS was not expensive.
C++ Emulator has no access to the internet.	Have to port the application to the phone in order to test and this is time consuming.	When the phone gets docked the code added has to be a substantial addition in order to manage time constraints.	Check how often the phone needs to be docked.	Try to write modular code in order to detect errors quickly.	The docking of the phone was a problem throughout the project. It was very time consuming to dock each time testing had to be done.

One member falling ill or dropping out of the project.	Other members of the team cannot proceed with their individual projects and the project could fail.	Each project member must be sure that they can test their components without the code of the other project members.	Team members must communicate about their health, academic progress and progress made in the project.	Each member must test not only with other project members' code but also with other programs that they have written or gained access to.	No member fell ill or dropped out of the project.
Bad connection due to the use of GPRS.	Difficulty in testing whether the client is able to send to the database.	The WiFi network can be used instead of the GPRS network.	Check if the connection causes errors in the application that may hinder sending to the database.	Use WiFi to send sightings and use GPRS only for certain test cases.	Periodically throughout the project there was bad GPRS connectivity. WiFI was then used as a substitute.
Difficulty of team members in trying to understand the current CyberTracker code, architecture and database.	Team members unable to proceed with their project component. The project could fail.	Whenever each team member makes progress with understanding a component of the CyberTracker system they report to the other team members to help them understand. Also, contacting the developer of CyberTracker to get advice.	Keep a written record of how different parts of the CyberTracker works and ask other team members for help.	Daily feedback from all group members as to their progress with understanding the existing CyberTracker system.	The team had great difficulty in understanding the CyberTracker code as it was not well-documented and commented. After emailing the developer who worked on CyberTracker many times some progress was made. He was only able to email once in a while so progress was slow.

Table 3: Mitigation and Monitoring of project risks.

4. Design

This section contains the overall design of the system. The format of the XML messages sent from client to server is discussed. The schema of the database that stores all the data that the client sends is given. The architecture of the entire system is also shown.

4.1 XML Message Formats

As mentioned above all messages sent from the client to the server and vice versa will be sent as XML messages. These XML messages will be well-formed messages. Messages received from the client will be checked for well-formedness. All messages received that are not well-formed will be rejected and an error message will be sent back the client.

Five different XML messages have been proposed. These are:

- Data messages which are either sighting data or waypoint data.
- Acknowledgement messages used to send to the client when a data message has been received.
- CTS messages used to notify a client that a server is ready.
- RTS messages used to send to the listener when a client wishes to send data.
- Error messages that is sent when an error has occurred on either the client or server side.

Each XML message contains a tag called <type>. The type tag can contain either “Sighting”, “Waypoint”, “ACK”, “CTS”, “RTS” or “ERROR”.

Figure 6 shows the XML format of a sighting data packet. It contains various fields that are used by the existing CyberTracker application. The `<sightingID>` tag is used to uniquely identify a sighting. The `<GPS>` tag contains all the data that make up a GPS reading of where the sighting was recorded. This field can be empty if no GPS was taken with the sighting. The `<Sighting>` tag contains all the data that makes up a sighting. A sighting is made up of a list of attributes. Each attribute has an element identifier and a value. The element identifier refers to the element that was recorded, for example a rhino, and the value refers to the data attached to this entity, for example the number 9 being the amount of rhinos spotted. The `<TimeStamp>` tag contains the date and time the sighting was recorded.

```
<cyber>
      <type>Sighting</type>
      <sightingID>12345-23-23-45</sightingID>
      <GPS>
            <Latitude>33</Latitude>
            <Longitude>33</ Longitude>
            <Altitude>33</ Altitude >
            <Accuracy>10</Accuracy>
            <Quality>10</Quality>
      </GPS>
      <Sighting>
            <Attribute>
                  <ElementGuid>12345-12-12-13</ElementGuid>
                  <ElementId>12357-15-12-67</ElementId>
                  <ScreenId>500</ScreenId>
                  <Value>5</Value>
            </Attribute>
      </Sighting>
      <TimeStamp>2006/10/31:10:28</TimeStamp>
</cyber>
```

Figure 6:A sighting data XML message. This XML message is what is sent from the client to the server when a sighting is sent. It contains various fields used by the current CyberTracker system.

Figure 7 shows a waypoint data XML message. It is the same as the sighting data message but has no `<Sighting> tag`.

```
<cyber>
      <type>Waypoint</type>
      <waypointID>12345-23-23-45</waypointID>
      <GPS>
            <Latitude>33</Latitude>
            <Longitude>33</ Longitude>
            <Altitude>33</ Altitude >
            <Accuracy>10</Accuracy>
            <Quality>10</Quality>
      </GPS>
      <TimeStamp>2006/10/31:10:28</TimeStamp>
</cyber>
```

Figure 7: A waypoint data XML message. This XML message gets sent from the client to the server when a waypoint is sent.

Figure 8 shows an acknowledgement message. It contains an `<id>` tag which has the sightingID or waypointID that has just been updated in the database.

```
<cyber>
      <type>ACK</type>
      <id>5</id>
</cyber>
```

Figure 8: An acknowledgement XML message. This XML message gets sent to the client once The database has been updated with a sighting or waypoint.

Figure 9 shows a RTS XML message. It contains a `<size>` tag that how many packets the data to be sent is.

```
<cyber>
      <type>RTS</type>
      <size>5</size>
```

Figure 9: A RTS XML message. This message gets sent from the client to the listener when the client wishes to send a sighting or a waypoint.

Figure 10 shows a CTS XML message. It contains a `<port>` tag which contains the port number that the server that is going to receive the client's sighting or waypoint data.

```
<cyber>
      <type>CTS</type>
      <port>5475</port>
</cyber>
```

Figure 10: A CTS XML message. This message gets sent from the listener to the client once a RTS has been received by the listener.

4.2 Database Organisation

In order to decide on how to implement the database component of the project the manner in which the existing CyberTracker system had to be observed. It was decided that the design of the CyberTracker database would be the same as the design of the existing CyberTracker database. These databases will store the same type of data so there was no reason to reinvent the wheel.

CyberTracker currently uses Microsoft Access for their databases. With CyberTracker you are able to load different sequences onto a mobile device. These sequences allow users to collect any type of data. For example, one user could have one sequence that they use to capture plant information and one that they use to capture animal information. For every sequence that is created a Microsoft Access database is created which will contain all the data that the user docks when using that sequence. For the purposes of this project only one database was needed that would serve as a central storage point. So, for example perhaps a team of geo-scientists decide to collect data in the field, the data collected by each scientist will go to one database. This will allow the integration of all team observations.

As mentioned in an earlier chapter the current CyberTracker system has two tables in its databases called Sighting and Waypoint2 which is used to store sighting and waypoint data respectively. The database used in this project has the same tables. Figures 11 and 12 shows the schema of the Sighting and Waypoint2 tables. The sighting data is saved in the Data and BigData columns of the database. It was decided that XML would be stored in this column in order to save all the data needed to store a sighting. The data needed has been shown in the XML message format shown in Figure 6.

Id	UniqueId	DeviceId	DateCreated	DateCurrent	Deleted	CheckedOut	Data	BigData

Figure 11: The schema of the Sighting table in the database.

Id	UniqueId	DeviceId	DateCreated	DateCurrent	Deleted	Latitude	Longitude	Altitude	Accuracy

Figure 12: The schema of the Waypoint2 table in the database.

4.3 System architecture

Figure 15 shows the architecture of the entire CyberTracker system. A C++ client connects to a Java application via GPRS. The three-tier Java application then updates the database.

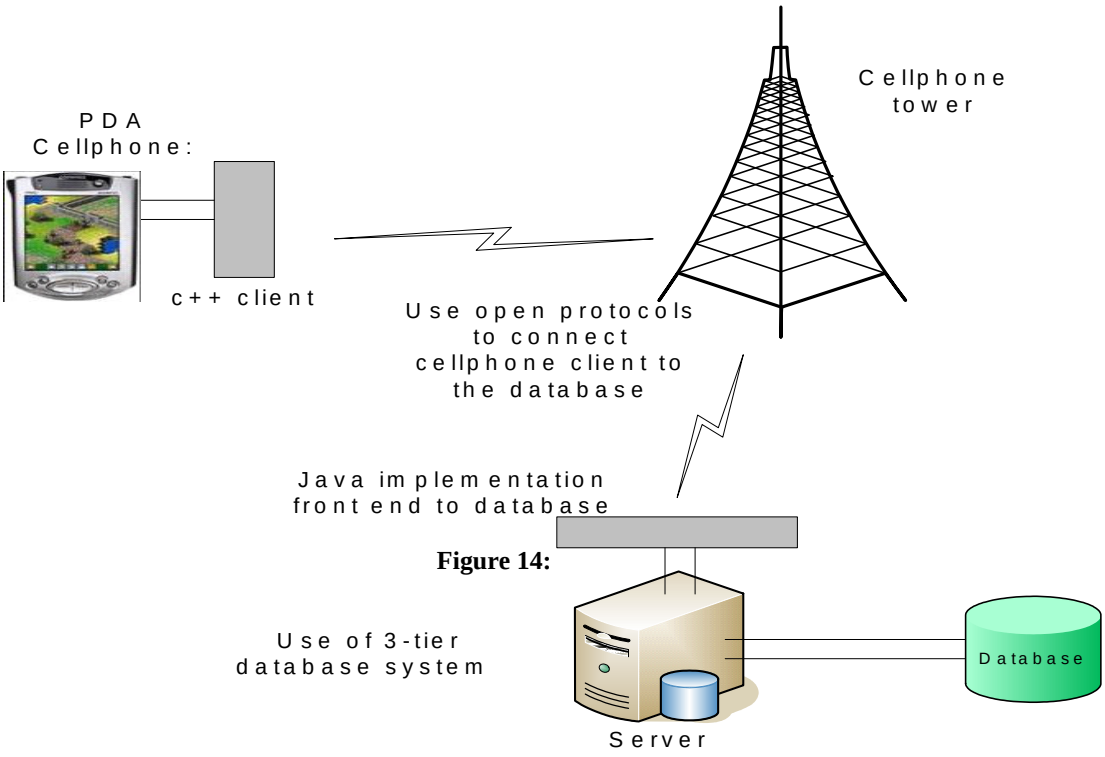

Figure 14:

Figure 13: The system architecture diagram showing all components of the system.

5. Implementation

This section contains a discussion of various technologies chosen to implement the project. A description of how the implementation developed during each iteration is also given.

5.1 Implementation Decisions

Various decisions have been made concerning the implementation of this project. A great deal of thought was put into deciding which technologies would be appropriate for this project. Decisions were made by considering the advantages, tradeoffs and suitability of the technology in question.

5.1.1 Using TCP for transmission of data

TCP was chosen as the channel to send all data. The project team felt that TCP (Transmission Control Protocol) is more suitable than UDP (User Datagram Protocol) for this project.

TCP provides services that UDP does not. Using TCP removes the need for the development team to implement these services. Such services include data checksums, timeouts and retransmits, duplicate detection, flow control and sequencing. The mobile client should contain as little overhead as possible, so not having to implement these services reduces overhead in this respect. These services also make TCP more reliable than UDP.

5.1.2 Using EJB to handle many clients updating the database

The choice to use EJB and a listener has already been mentioned but there are further reasons why these technologies was used.

With the extension of CyberTracker to allow for the sending of sightings using GPRS there is the possibility of many new users using the CyberTracker application. For example, bird watching societies could use CyberTracker to keep a record where birds were observed. CyberTracker is adaptable in terms of the type of data that can be captured. Users can choose to utilise the application to collect any type of data. It is therefore feasible to allow for the possibility of an increased user base.

With the possibility of an increase in users the extended CyberTracker system should allow for scalability. Scalability should be catered for in terms of allowing the system to handle many simultaneous user requests. Scalability should also be catered for in terms of allowing the system to handle many new users.

Using a three-tier client-server architecture is a means of allowing for scalability. In terms of CyberTracker a three-tier could consist of a client, a middle layer and a database. Clients communicate with the middle layer instead of directly communicating with the database. The middle layer can then contain the application logic. This will allow the client to have less overhead which is suitable for the purposes of this project as the client is a mobile application with possibly limited resources.

In order to implement a middle layer the use of middleware was considered. COM+ was considered as it is written in C/C++, the same language as the client application. COM+ is however proprietary software and CyberTracker aims to being completely opensource. Enterprise Java Beans was then explored as an option. EJBs use Java so this would mean that the server and the client would be implemented in two different languages. EJB was however chosen as the only free middleware that could be written.

When using container managed EJBs all transactions can be handled by the container. Each method of a bean will be treated as a transaction. In order to allow for container managed transactions the transaction field within the container must be set to “required”. When using Jboss application server which is the application server used in this project, setting the <trans-attribute> field of the <container-transaction> in the <assembly-descriptor> field in ejb-jar.xml to “required”does this. Allowing the container to handle and manage transactions allows for simplicity in terms of implementation. Because of using EJB the need to write commit protocols is not needed. The container can handle synchronisation. This means that the application programmer will not need to implement threads to handle many users accessing database information.

5.1.3 Listener to handle simultaneous client requests

With the decision to use EJB a means was needed to allow many clients to make requests at the same time. If many clients are allowed to make requests at the same time then many clients can access the EJBs at the same time. This is needed to allow for scalability.

When using EJB there are two ways in which clients could access the EJBs. One way is to build a web interface and the other way is to build a client application. Since there is no need for a web interface in this project as this project is concerned with the backend, a client application had to be used.

The Server class would act as the client application accessing the EJBs. It was decided that one client would be handled by one server. The addition of a listener would be a way for each client to be assigned to a server. This approach was then decided upon. Each client sends a RTS to the listener running on a certain port. The listener then spawns a server and the server sends a CTS back to the client. This CTS contains a random number which will be the port that the client and server will have any further communication through.

A mechanism was then needed for the listener to spawn servers. The initial approach was for the listener to create a new process for each server. This was further explored and it was decided that the use of threads would be more appropriate. There were two reasons for this. The first reason was that if the server crashed at any time before the server started operating on the random port then the listener would also crash. The second reason is that threads share the same resources whereas each process requires its own resources. This makes threads more lightweight than processes.

5.1.4 Using MySQL as a database

The existing CyberTracker system uses Microsoft Access databases. The logical choice for the database used in this project would thus be a database of the same type. Microsoft Access is not however tailored for multi-user applications. A database management system that is web-based is more suited to this project as such a database is likely to be more scalable. It was thus decided to use MySQL as it is freely available and suits the need for CyberTracker to go completely open source.

5.2 Development of the implementation

This project was iterative in nature with a working implementation being available at the end of each of four iterations. The analysis and design was also reviewed in every iteration ensuring that all aspects of the project were iterative.

5.2.1 Iteration 1

Iteration 1 was a short iteration and was aimed allowing simple communication between the client and server. At this stage the client was a C++ application as the mobile phones has not arrived yet.

The server was a Java application which ran a Java ServerSocket on a specified port and waited until a connection with a client was established. Once the connection was established the server tried to read from the socket for a pre-defined amount of bytes. This was a blocking read so the server would wait until it had anything to read from the client.

The message “Hello World” was sent from the C++ application to the Java server. There were problems initially with converting the messages into the correct format once being sent, but these were sorted out quickly.

A Java client was also developed to ensure testing could go on without the participation of the mobile application.

5.2.2 Iteration 2

Iteration 2 was concerned with transmitting XML messages from the client to the server. The server did not change much in this iteration except for the fact that it could accept an XML message and return one to the client. Work was done on the Message class to ensure that the XML messages that were received were well-formed. The message class also provided a means to easily create XML messages to send back to the client. With methods like createACK(String id). This message takes the sighting id or waypoint id of sighting or waypoint data and creates a well-formed XML acknowledgement message (in the format shown in figure 8). In order to ensure that the received messages were parsed with a DOMParser. The DOMParser provided a simple way to check for well-formedness and to extract data out of the messages was received.

The Java client was developed further to be able to send XML messages.

In this iteration extensive work was done in trying to understand how the existing CyberTracker database works. There was uncertainty when it came to understanding how sighting data get stored. This was not deciphered in this iteration. The project team had also just received the existing CyberTracker code and was trying to understand how the code works. Unfortunately the code had few comments and no documentation. This consumed a great deal of the project group's time. The group started emailing the developer that wrote CyberTracker and got replies once every day or second day. Progress in understanding the existing code was slow at this point.

5.2.3 Iteration 3

Iteration 3 was the iteration in which the most progress was made. The goal of this iteration was to be able to send waypoints and sightings to the server. Once the server received the waypoint or sighting the database had to be updated.

The EJBs were written at the start of the iteration. Three EJBs (CTSession, Sighting and Waypoint2) were written and Jboss 4.0.4 was chosen as the application server. The Server class was extended to send CTS, ERROR and ACK messages. The project team decided on a packet length of 1024 bytes. With this size one sighting would usually be transmitted in two packets. The server first tried to handle only one packet. That was tested and it was working so the server then went on to accept a sighting or waypoint broken up into many packets.

With allowing for the receiving of many packets the protocol had to be adjusted. After the server received every packet an acknowledgement was sent to the client. Once the server received the last packet it checks the message for well-formedness and it also checks for certain fields. It checks the type tag. Depending on what kind of message was received the server will try to update the corresponding table in the database by using the CTSession bean. If the database was successfully updated then another acknowledgement was sent to the client. The server's work is then complete. With the receipt of this acknowledgement the client would know that the data was successfully inserted into the database.

The implementation of the Server was tested with the Java client which was updated to be able to receive acknowledgements and send data in the format of the agreed upon sighting and waypoint formats (figures 6 and 7).

At first the database used was HyperSonic, Jboss's default database. Some time was spent trying to get a Microsoft Access database to be updated in order to stay compliant with what the existing CyberTracker uses. This attempt was cancelled when considering that CyberTracker wishes to be totally open source and considering that the type of database used was not important as long as many users could access it at one time. Additionally Jboss did not have sufficient documentation on allowing it get configured for a Microsoft Access database. As mentioned above MySQL was chosen as the database to be used as it is web-based and freely available. Jboss was then configured to use MySQL and all the data received by the client would go to a MySQL database.

Work was also done on a small class containing error codes, called ErrorCodes. This class maps a string containing an error code for a certain type of error. This class is used when error messages

are sent to the client. Error messages are sent when incorrect packets have been received, or when data could not be read as it timed out or when the database could not be updated. The Message class was also updated to parse sightings and waypoints that were received so that they could be inserted into the correct fields of the database. At this point any message that was sent was created using the message class to ensure a well-formed XML format.

The server was then checked with the mobile client and many problems were run into while trying to get the two applications to work in sync. There were problems with corrupt data being sent from the client to the server. There were also problems with getting the protocol to work properly between the client and the server. The protocol was then implemented correctly on both the server and mobile client side. There were problems with the mobile client connecting to the server. Connecting to sockets were not always working and sorting this out was very time consuming as debugging on the client side is very slow as the phone needs to be docked to test. These bugs were sorted out and sending a sighting or waypoint from the mobile client to the server worked successfully with the updating of the correct table in the database.

5.2.4 Iteration 4

Iteration 4 focussed on getting a listener to spawn servers in order for many clients to be handled at one time. Initially the approach to spawning servers was to create a new process for each server. It was found however that when a server crashed before sending the CTS it would cause the listener to crash. The listener should not crash as it must continuously handle client requests. One way that the for the server to crash is when the client exits and the socket is closed on the client side. This causes an IOException as the connection between the client and server is reset. This IOException cannot be handled in Java so the crashing of servers was presenting a big problem. After much consideration it was decided that threads would be used for the servers. When one server thread crashed it did not cause the listener to crash.

The listener and server method was tested on the Java client and the sending and receiving between the client and server was working correctly. It was not however working with the mobile client. It was then decided that that the server would run on a random port and this functionality was put into the server. The server runs on a random port and the port is checked to see if it is busy before it starts to run on that port. This approach worked with the mobile client but there were still some problems as with the sending and receiving of data. The data would sometimes not be received by the client. After some time these problems were sorted out and sending and receiving worked properly.

At this time a converter mechanism was being worked on. This converter method would take the data in the MySQL database and convert it into a format that could be read by the CyberTracker desktop application. The converter would write all the data in the database to file. A separate application would read from this file convert the data into the required types and insert into a Microsoft Access database. This would allow the database in the MySQL database to be viewed using the CyberTracker desktop application. Unfortunately only the first step of this process was done. Currently the data from the MySQL database can be written to file but time ran out before the other steps could be implemented.

6. Testing

6.1 Iterative Testing

Testing was done during each iteration and at towards the end of each iteration. During the iteration the system was tested with a Java client. The Java client was developed to accommodate the progress of the system. For example, if the server was able to receive XML messages and parse it then the client was able to send valid XML messages. The client in this case would also send invalid XML messages in order to see if the server could detect and recover from it. Both white box and black box testing was done in each iteration. White box testing was done to verify that each method was working. Black box testing was done when a certain function of the system had to be tested.

Towards the end of each iteration the system was tested with the client running on a mobile client. As each test was done the errors were recorded and were then fixed before the next iteration was begun.

6.2 System testing

The system was tested using both a Java client and the client running on a mobile device. The Java client was used to test the server's error recovery ability and to test whether the server could handle many clients.

6.2.1 Error Recovery Ability

In order to test the server's error recovery ability a Java client was developed that simulated different error situations. These included the client sending many incorrectly formatted messages, the client taking too long to send a message and the client sending data when the EJBs were not deployed and able to update the database.

In order to test whether the server could handle receiving incorrect messages, a number of incorrectly formatted messages were sent from the Java client to the server. Figure 14 shows an incorrectly formatted XML message that was sent to the server. This XML message had an incorrect `<type>` tag. When the server received this XML message it sent an error message to the client. This is because the <type> tag within a data message sent from the client must have a value of "Sighting" or "Waypoint".

```
<cyber>
   <type>wrong type</type>
   <sightingID>123456</sightingID>
   <GPS>
         <Latitude>30</Latitude>
         <Longitude>23</Longitude>
         <Altitude>50</Altitude>
         <Accuracy>10</Accuracy>
         <Quality>100</Quality>
   </GPS>
   <Sighting>
         <Attribute>
               <ElementGuid>12345-12-12-13</ElementGuid>
               <ElementId>12357-15-12-67</ElementId>
               <ScreenId>500</ScreenId>
               <Value>5</Value>
               </Attribute>
   </Sighting>
   <TimeStamp>15/10/2006</TimeStamp>
</cyber>
```

Figure 14: An XML message that has an incorrect <type> tag. This message was sent from a Java client to the server in order to test the server's ability to handle incorrect messages.

Figure 15 shows a RTS message that was sent to the Listener. This RTS message had an incorrect `<type>` tag. When this message was sent to the Listener it did not send a CTS message to the client as a RTS message must have a `<type>` value of "RTS".

```
<cyber>
      <type>Not an RTS</type>
      <size>1</size>
</cyber>
```

Figure 15: An RTS XML message that has an incorrect <type> tag. This message was sent to the listner in order to test whether the listener could handle incorrect messages.

In order to test whether the server could handle completely incorrect messages a message containing "Incorrect message" was sent to the server from the Java client. When this message got received by the server, it crashed as a NullPointerException was encountered. This was because the server was expecting a message that had a `<type>` tag. This was fixed and when the client sent this message the server sent an error message back to the client.

The server's error recovery was tested further by sending a sighting that had no sighting id. This message is shown in figure 16. When the server received this message it sent an error message back to the client.

```
<cyber>
    <type>Sighting</type>
    <GPS>
        <Latitude>1000</Latitude>
        <Longitude>23</Longitude>
        <Altitude>50</Altitude>
        <Accuracy>10</Accuracy>
        <Quality>100</Quality>
    </GPS>
    <Sighting>
        <Attribute>
            <ElementGuid>12345-12-12-13</ElementGuid>
            <ElementId>12357-15-12-67</ElementId>
            <ScreenId>500</ScreenId>
            <Value>5</Value>
        </Attribute>
    </Sighting>
    <TimeStamp>15/10/2006</TimeStamp>
</cyber>
```

Figure 16: An XML sighting message that has no sighting id.

The client sent a valid sighting to the server when the EJBs needed to update the database was not deployed. Once the server received the sighting it sent an error message to the client as the EJBs could not be located and thus the database could not be updated.

A further test of error recovery was when the client takes to long to send a message once a RTS was received. This may happen because the client could have lost its connection or the the mobile device could have switched off. To simulate this situation the Java client sent a RTS message to the server. The server then sent a CTS message to the client. The client then slept for twenty-one seconds. The server then sent an error message to the client as the client took too long to send a data message.

6.2.2 Handling many clients

The server should be able to handle many clients making requests and trying to update the database at the same time. To test whether the server could handle many clients a Java application was developed that created many clients. Ten clients were spawned that sent requests at intervals between 1.5 and 5 seconds and less. To check whether the database was updated for every clien the program that writes all the contents of the database to file was used. Once the clients were done sending their data the contents of the file was observed. It was found that the database was updated correctly. The server is thus able to handle many clients.

6.2.3 Testing with the mobile client

The server was tested extensively with the mobile client by sending various sightings from the mobile device to the server. The server was able to handle the mobile client's requests and data. This was verified by checking the contents of the database by running the program that writes all the contents of the database to file. This file was checked every time data was sent from the mobile client. To check if the server received a valid message the contents of the sighting was looked at. The value fields in the sighting was checked to see if it corroborated with what was sent from the client. Many inconsistencies between the mobile client and server were discovered in iteration three. The implementation of the protocol for communication between the mobile client and server was worked on extensively until all inconsistencies were removed. Once this was done it was found that the correct database table was being updated every time the sighting was sent. It was also found that the server was receiving valid data by checking the value fields.

There were times when the mobile client did not successfully transmit data to the server. In these cases it was verified that the server behaved in the correct manner by sending an error message to the client.

7. Findings

The analysis and design of the system proved to be useful ways to provide a basis for the implementation of this project. The software engineering artefacts developed during the analysis phase of this project provided a means to extract out the functionality that was required of the system. The design of the system provided a means to place all the components of the system into perspective.

Using iterative development enabled the project team to build many versions of the system which were improved and added on in every iteration. Updating the analysis and design of the system in each iteration also ensured that the final system developed was appropriate.

It was shown from testing that the technologies used were appropriate to implement the CyberTracker server. The listener/server approach proved to be a way to successfully handle many client requests. Using EJBs provided a way to update the database, handle synchronisation and allow for transactions in a manner that was simple to implement. Using XML was not only a means for the mobile client and the server to communicate in a language-independent manner but also a means to effectively extract data. The data received could be retrieved by using a DOM parser. Using the DOM parser also aided in checking if the received messages were in the correct format.

Many of the risks that were mentioned in the risk analysis became a reality in this project. Trying to manage these risks was a challenging task for the team members. It was due to these risks that the problem of querying the database and sending back relevant results to the database was not implemented. The aims of providing a means to handle client requests and a means for a mobile client to update a remote database were met.

7 Conclusions

7.1 Future Work

This project could be extended in a few ways in order to add value to CyberTracker and it users. The first extension that needs to be done is to allow the users of the CyberTracker application to be able to view the data in the MySQL database. It is unfortunate that this was not done in this project but time ran short as the project team was faced with many risks and problems associated with understanding the current system. An estimated time of one week would be needed to allow for this functionality to be implemented.

This project has dealt with sending information to a remote database. Future work could involve querying the database. This would not be a difficult task on the server side as the data can easily be extracted from the database using EJBs. This will however require some new interfaces on the mobile client side. One example of where querying could be particularly useful is for users to view where they were on the previous day. This will require querying the Waypoint2 table which could easily done by creating a function to return all waypoints that occurred on a particular day.

7.2 Assessment of the project

The project plan was adhered to in terms of when the different iterations started and ended, but what was completed in each iteration was not what was planned at the beginning of this project. There were various reasons why the progress of this project was not what is what predicted to be. Understanding of the existing CyberTracker system became a major stepping stone in this project. Unfortunately the existing system was scarcely documented and the organisation of the system had to be figured out by the project team. Although the project team did successfully understand the system after time passed, the progress in doing so did not happen in a fast enough manner. The team had to decide between understanding the current system thoroughly and developing a system that may not be suited to the current system but has more functionality. The team ended up choosing to develop a system that could integrate with the current CyberTracker system although all aspects of the functionality were not there.

Another risk that hindered the progress of the project was the difficulty with which the team members working on mobile application had with building the existing code. Only once this was done could the mobile client and the server be tested to work with one another.

The main project aims were successfully met. These were being able to handle many client requests and allowing a mobile client to update a remote database. The aim of querying the remote database was not met and would be a useful addition to the system if future work gets done on the system.