



SET 10101 Software Architecture Coursework

40280889 | Dominic Manderson

Table of Contents

Software Architecture Recommendations	3
Data Centred Repository.....	3
RMI (Remote Method Invocation)	4
Aspect Oriented Software Architecture (AOSA).....	5
KwikMedical Final Recommendations	7
Repository and RMI.....	7
Prototype Design.....	9
Evaluation.....	13
Future KwikMedical Developments	15
References.....	16

1 Software Architecture Recommendations

1.1 Data Centred (Repository)

KwikMedical relies heavily on having a persistent data store In order to achieve its fundamental functionality of storing data such as the medical records of patients. Additionally, KwikMedical is a distributed system, meaning that components should be spread across multiple machines which maintain a network connection capable of sending and receiving messages across the system. To create a distributed system, a network communication technology such as RMI needs to be included. Therefore, one of the possible software architectures proposed is a combination of both of these technologies, which then becomes known as heterogeneous, since it makes use of multiple architectural styles.

The data centred architectural style consists of two different types; blackboard and repository. The blackboard style allows multiple clients to work on a central data store which remains active – a good example of this can be an artificial intelligence knowledge base. The repository architectural style better fits the specification of the KwikMedical system, and is therefore the recommended style of the database. A repository styled data centre uses a passive data store which may be queried by a client at any time. The components found in this architectural style are the database itself and the clients (telephone and ambulance operators) which access it. Connections are made between these components when a client queries the database via product protocol, using a technology such as JDBC (Java Database Connectivity) which is a method of information exchange in Java. A client may have only one connection to the database. With the connection, a client may pull or update records at any time across a distributed system.

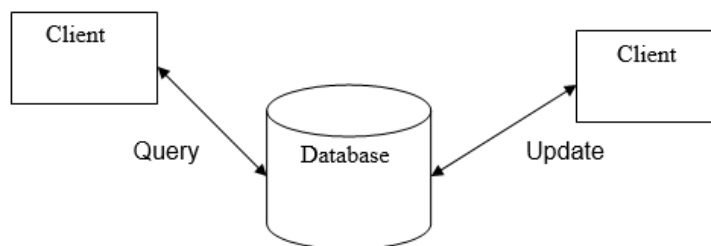


Figure 1 - Repository database model (Liu, Data Centred Architectural Styles)

1.2 RMI (Remote Method Invocation)

KwikMedical is a distributed system, and therefore requires communication across multiple machines. When using RMI, an interface must be used in order to specify the possible interactions between processes - the interface itself only specifies details of methods and variables. With this interface, a level of abstraction is achieved, which makes it easy to change the implementation of a procedure call to suit changing business requirements.

RMI transfers data from client(s) and a server with *stubs*. A client will have access to a local stub, which it can use to marshal and send a request through an active server onto an

application connected to that server. When a reply is received, the stub then un-marshals the response and sends them to the client. For the client, it seems like the method being called is local, as the complexity is hidden with middleware. With this, a client operator can process all of the patient information seamlessly from their given workspace.

The server which is connected over a network contains a dispatcher which selects the server skeleton (stub) which was given in the request from the client. The arguments given from the client are then un-marshalled, and the procedure which was called may then run with the given arguments. Finally, the response is sent back to the client from the server side application - this functionality given with RMI allows a client to call a method and execute commands through a server which may not be on their local machine.

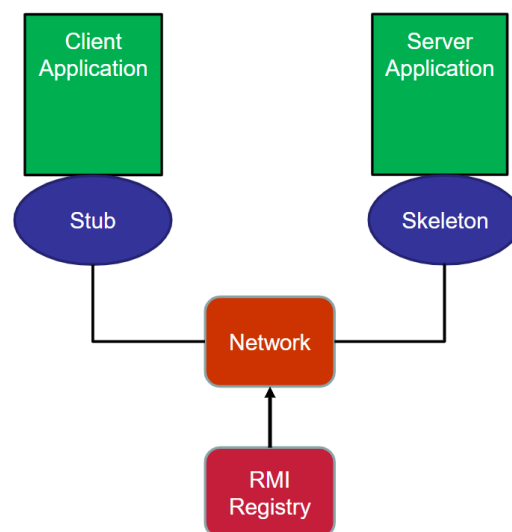


Figure 4 – Remote Method Invocation (Liu, Remote Invocation)

1.3 Advantages of database architectural style

The main benefit of using a database is having data which is secured on one machine that can easily be backed-up, or relocated onto a different machine. Additionally, many database tools exist which offer security features such as login systems and admin privileges to control an accounts access.

Another benefit is seen when discussing the standard agents which access a database, they may be scaled up and offer reusability since they do not need to directly connect to each other.

Finally, a database system can also improve in performance and efficiency to suit the needs of a project if they change in the future and require scalability. This can be done by spending capitol on a more powerful machine.

1.4 Disadvantages of database architectural style

Having data centralised may offer an advantage in security, but can also cause bottlenecks since the clients are all accessing the same data store. This could potentially cause the system to lack in performance and efficiency, which would only be fixed through distributing the databases or upgrading the hardware of the database.

The entire system is vulnerable due to a centralised database, meaning that if the database were to fail then the system as a whole would become useless. This situation can be avoided by using distributed databases to store data, which may also reduce bottlenecks.

The data integrity must be maintained, meaning that the data captured must be complete, accurate, and consist for example with the data structure expected. Without data integrity, the data captured becomes unreliable and irrelevant. There are many possible cases where data integrity can be ruined, such as human error.

2 Aspect Oriented Software Architecture (AOSA)

Aspect oriented programming has been introduced to help deal with large systems that contain many interconnected components, such as KwikMedical, where it may be difficult to deal with a change in requirements for the system. The problem that may occur is that changing one component may cause changes in the other components. This leads to a system which is difficult to maintain and lacks reusable components.

AOSA addresses this problem by introducing a new type of abstraction called an *aspect* – which is used with other types of abstractions such as methods and objects. AOSA maintains the separation of concerns, meaning that one element of a program should only focus on achieving one requirement, or concern. A concern is seen from the viewpoint of stakeholders, and could be for example the performance or security of a program.

Cross-cutting occurs when changing one component causes other components to also be changed. This could cause negative repercussions such as scattering, which occurs when one concern is implemented across multiple components in a system.

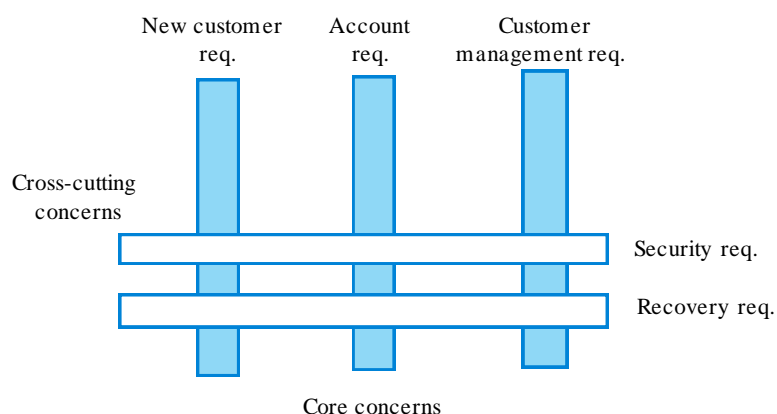


Figure 2 – Cross cutting concerns (Liu, Aspect Oriented Software Architecture)

An aspect is the implementation of a concern, such as security, which may be coded in a way that checks a user's credentials wherever needed – this achieves a reusable block of code which can be used across different components of the system.

Aspect oriented contains a core system which implements the main concerns, and this core system is surrounded by extensions which implement secondary and cross cutting concerns.

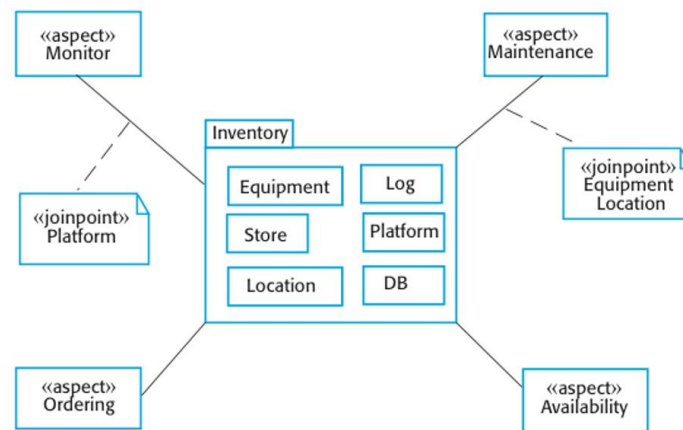


Figure 2 – example of core and extensions in warehousing scenario (Liu, Aspect Oriented Software Architecture)

In the core system of the design for KwikMedical, we could place the database to store patient records and callout details, and have this accessed by the clients (telephone operator and ambulance operator). We could use aspects for cross cutting concerns, such as authenticating a user. Since AOSA is an auxiliary architecture, it may easily become heterogeneous, using another architecture such as Client/Server to distribute the system over a network instead of being localised across one large system.

Client/Server is a software architecture for distributed systems where each computer acts as either a client or a server, where a client consumes a service and a server provides a service. For KwikMedical, a solution can be made which uses a web browser interface which connects to a web server, which then connects to a database. In this architecture, we identify that the client browser interfaces are components, along with the web server and database. The connection being made between the client browsers may be in a standard internet data transfer protocol, such as HTTP, and the KwikMedical application server can access the database with SQL.

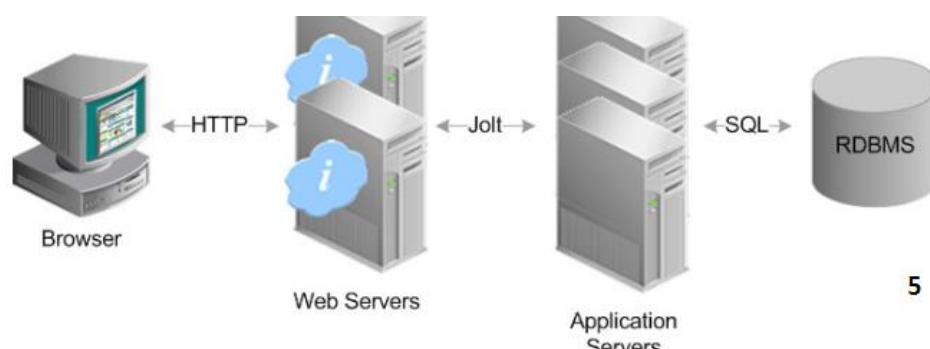


Figure 3 – Structure of a client/server system using web servers (Liu, Peer-Peer Systems)

This addition would make AOSA a good candidate for the KwikMedical contract, as the distributed system functionality will be achieved along with a centralised database. In this architecture, the components would be the user interface(s), the server interface, and the database, with a connection between these made with the RMI protocol, connecting based upon the RMI registry and data being transferred via the client side stub to the server side stub (these terms are explained further in section 2).

2.1 Advantages of Aspect Oriented Software Architecture

The main benefit with aspect oriented is the ability to separate concerns and write concerns into the code with aspects. This gives the advantage of reusable code, allowing aspects to be called multiple wherever needed which also makes the software solution smaller, and saves time on writing code.

The system is also easier to adapt, since you can change a specific aspect without changing the functionality of any other aspects. This also makes it easier to identify problems and fix them, making maintaining the system easier.

Also, aspect oriented is regarded as an auxiliary architecture, which can be paired seamlessly with other software architectures, such as Object Orientation, RMI, Client-server, etc.

2.2 Disadvantages of Aspect Oriented Software Architecture

Inspecting and testing a system which uses aspects is difficult, since there is no standardised testing available. There are several issues, such as how to test the aspect independent of the system, or how to specify aspects so that tests can be derived.

The problem of inspecting a system which uses aspect oriented is that there can be difficulty when reading the program since it can be compared to a web more than a linear document, this is due to not being able to tell where an aspect is executed just from the source code alone.

3 KwikMedical Final Recommendations

2.1 Repository and RMI

The architectural style chosen to satisfy the specifications required for the KwikMedical system will be heterogeneous, meaning that it will contain multiple different architectural styles. The styles which are recommended for the distributed system KwikMedical are RMI (Remote Method Invocation) and a repository styled database. The architecture which is displayed in the prototype will be best described as three tiered. The reason that this

architecture was chosen over aspect oriented was because we do not feel that the current demands of KwikMedical would make use of aspects as the system is not yet large enough. It is, however, a technology which could be implemented in the future when the system becomes larger and can benefit more from the addition. Furthermore, the use of client/server with a web server is not something that yet needs to be implemented, since there is no need to have a browser based user interface for KwikMedical at this time. Of course, this is an addition which could be added based upon changing business requirements.

The database is essential in this system since it satisfies the requirement of a client being able to record, store, update and retrieve the data of a patient or callout. The repository style has been chosen due to the database being passive, and only being accessed when a client makes a request to retrieve or insert data, instead of being constantly active as seen in the blackboard style.

The database will deliver a means of storing secure, validated, accurate data since the database itself may be accessible through any trusted SQL platform such as Oracle Express or wampServer. The security is ensured with a platform requiring admin privileges or login information in order to manually modify the structural integrity of a database. Additionally, with the use of RMI, only clients connected to the specific server may have access to the database.

Data integrity is also relatively simple to achieve, since restrictions may be placed on certain columns allowing it to only accept a specific type of input. Using a database will also allow scalability in the system, with the opportunity of investing in hardware in order to improve efficiency and performance at a monetary cost.

The architecture of the system itself will include three tiers. The first tier will be the clients, and act as a user interface which collects input from a certain client component (such as a telephone operator or a mobile device operator in the rescue ambulance). The second tier will be the server interface, which is implemented using the RMI protocol to process requests between a client and the data store whilst hiding any complexity from the clients. The third tier will be the database component, which is implemented using SQL and wampServer, and connected to by the server interface component, which connects via JDBC protocol.

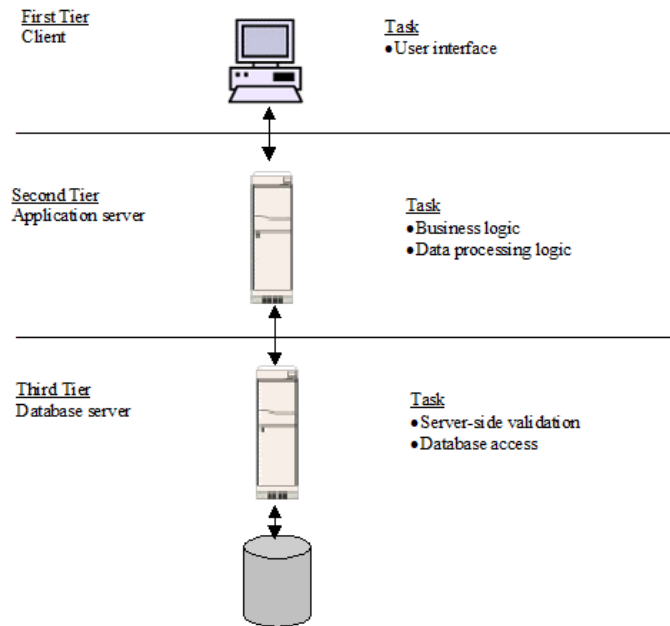


Figure 4 – Tiered architecture model (Liu, Data Centred Architectural Styles)

RMI is expected to be the protocol of information exchange which will allow client user interface components (telephone operator, ambulance operator) to make a connection to the server interface component which will be KwikMedical. This connection will allow data such as patient records to be passed by the stub of the client onto the stub of the server interface, which will then access the database and perform an operation using the data passed from a client user interface. This architecture will mean that the repository database can only be accessed through the server side application KwikMedical.

The database will be hidden behind the server, and clients may run independently of each other, but only be able to start their respective client application if the KwikMedical server is up and running. The component diagram for the proposed architecture can be seen in the following section, along with the prototype design. For the prototype, RMI will be implemented using the RMI framework in java, this will give KwikMedical the functionality of a distributed system, since object orientation is also being used in the prototype, and RMI supports distributed objects plus the passing of data between these objects.

4 Prototype Design

3.1 Class Diagram

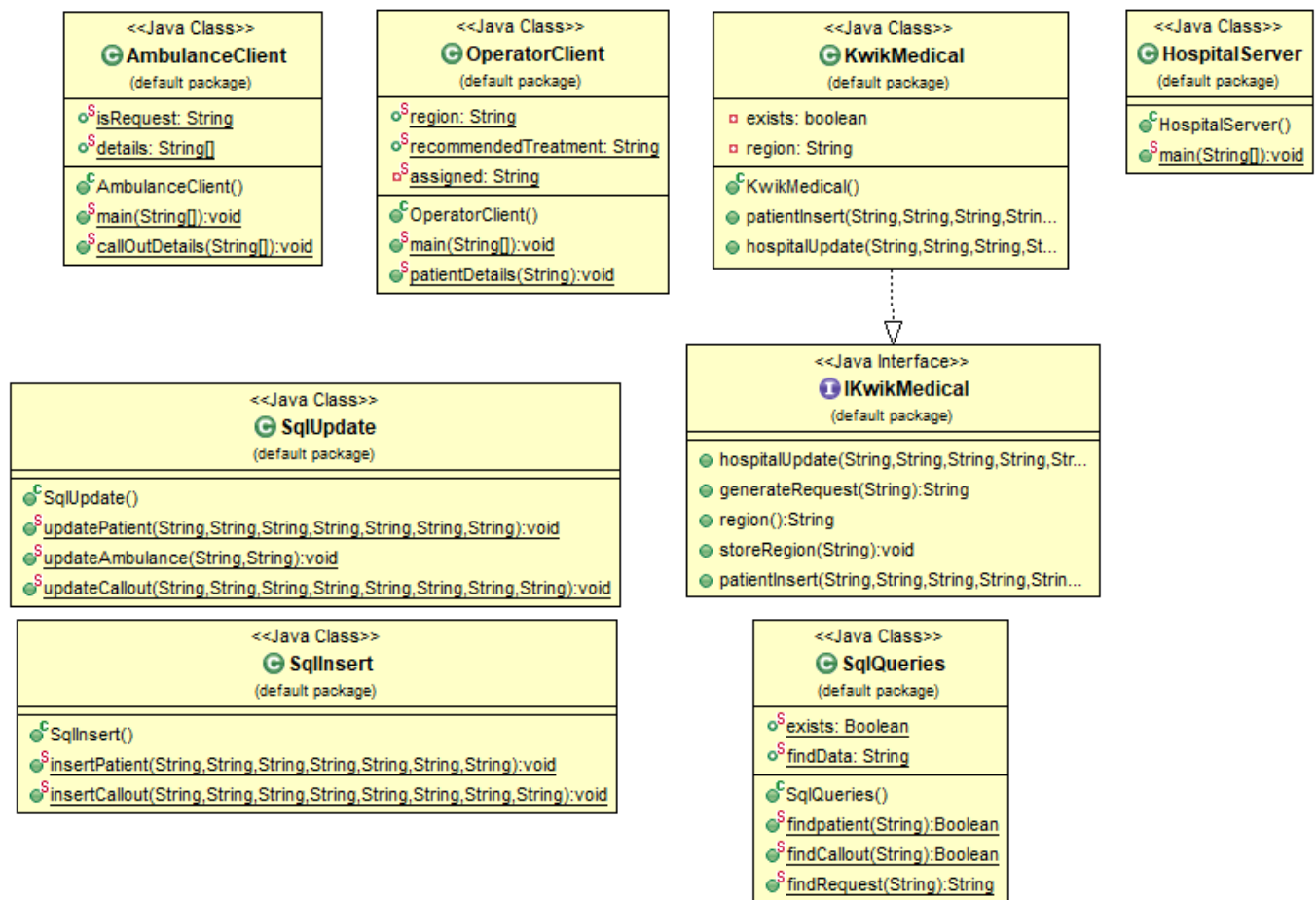


Figure 5 – Class diagram generated in Eclipse

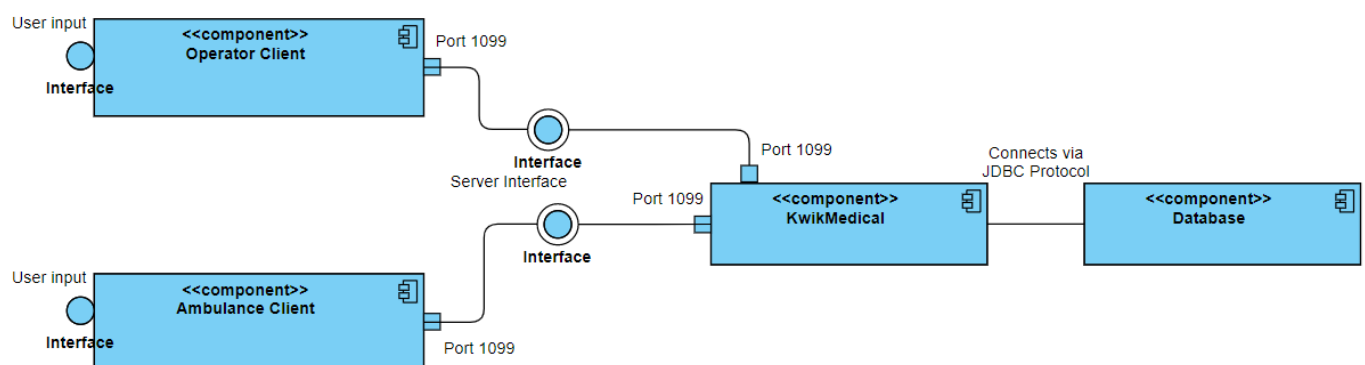


Figure 6 – UML component diagram of system architecture

3.2 Tools used

The prototype will be programmed using the Java programming language and the Eclipse IDE.

Object Orientation will be the chosen method of programming, to promote reusability in code and hide complexity from a user.

The Java RMI framework will be used to display that KwikMedical is a distributed system, and to create a server which will enable a client/server approach.

The JDBC connection protocol will be used to connect to the database server (wampServer) from a client's machine creating an API.

wampServer and phpMyAdmin will be used to host the database.

3.3 Graphical User Interface (GUI) and Prototype flow

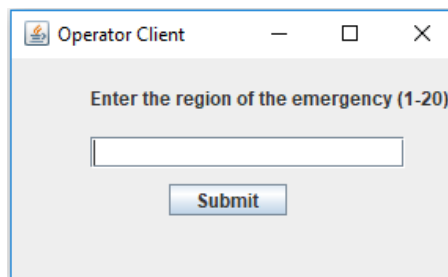


Figure 7 – GUI interface to receive region of emergency from telephone operator.

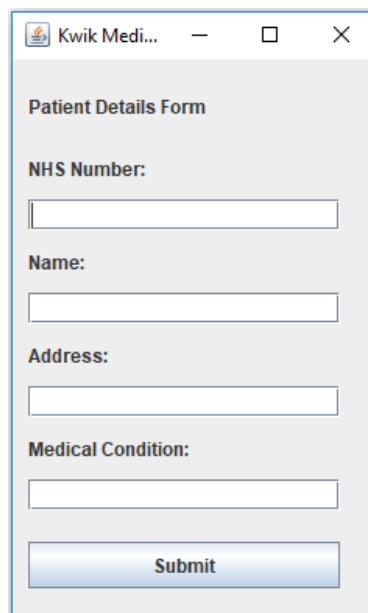


Figure 8 – GUI interface to receive patient details from telephone operator.

Figure 9 – GUI interface to receive ambulance operators region to receive the recent emergency indecent in that region

Figure 10 – GUI interface to receive ambulance operators callout details and update the database with them when submitted to hospital

3.3 Database structure

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	PatientNHSNo	varchar(10)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 2	PatientName	varchar(100)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 3	PatientAddress	varchar(100)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 4	PatientCondition	varchar(100)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 5	RecommendedTreatment	varchar(50)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 6	Region	varchar(20)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 7	AmbulanceAssigned	varchar(3)	latin1_swedish_ci		No	None			Change Drop More

Figure 7 – Patient records table structure and parameters

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	NHSNo	varchar(10)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 2	PatientName	varchar(100)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 3	PatientCondition	varchar(100)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 4	Date	varchar(10)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 5	Time	varchar(5)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 6	LocationOfAccident	varchar(200)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 7	ActionTaken	varchar(200)	latin1_swedish_ci		No	None			Change Drop More
<input type="checkbox"/> 8	TimeSpent	varchar(4)	latin1_swedish_ci		No	None			Change Drop More

Figure 8 – Callout details table structure and parameters

5 Evaluation

The prototype achieves the basic functionality which KwikMedical is supposed to have, with the use of a repository database and RMI. With the prototype, a headquarters operator may simulate receiving a call and launch their end of the program. The operator is then prompted for the region of which the emergency has taken place, so that this may be recorded and an ambulance rescue request sent to that regions hospital. Then, the operator is prompted to enter the patients' medical information into a form which is seamlessly sent via RMI to the server side KwikMedical application, which then accesses and inserts the patient's information into the repository database located on wampServer.

For the rescue request, an ambulance operator may enter their current region into the mobile device and be updated with the most recent request in their region, which will send the patient's details and a recommended medical treatment (if there is no request, none will show). The ambulance operator may then update the mobile device in the ambulance with the details of the callout, which are automatically sent back via the server side application and stored in the database online. Currently, this functionality is sufficient, but there are some additional features which will provided when we have secured the contract to build the KwikMedical system.

This functionality has been tested and delivers in the relevant areas which can be seen in the Moscow evaluation.

4.1 Moscow evaluation of prototype

ID	Priority	Requirement	Complete?
1	Must Have	Operator enters input of patient name, NHS number, address and medical condition into system.	Yes
2	Must Have	Passive repository which may be queried by clients.	Yes
3	Must Have	Verify if patient exists and update existing record or add new record (avoids duplication in database).	Yes
4	Must Have	Assign patient a hospital, and generate request for ambulance rescue.	Yes
5	Must Have	Work out the best way of helping the patient.	Yes
6	Must Have	Send request to allocated ambulance in region containing patient info.	Yes
7	Must Have	Ambulance updates callout details into the hospital system.	Yes
8	Must Have	RMI Server must be running in order to access the database.	Yes
9	Should Have	Use a GUI to take user input (looks clean).	Yes
10	Should Have	Prevent database duplication (based on NHS number primary key).	Yes
11	Should Have	Operator client side and Ambulance operator client side work independently.	Yes
12	Could Have	Connect the patient records and call out details table with a request table.	No
23	Could Have	Distance calculations to determine a closest ambulance.	No
14	Won't Have	Use GPS to assign a vehicle more efficiently.	No
15	Won't Have	Distributed databases.	No
16	Won't Have	Testing with real equipment such as an ambulance and patients.	No

4.2 Future developments to KwikMedical

4.2.1 Input validation

One of the largest flaws with the current prototype is that there is currently very little input validation, the only validation that exists is on the database columns themselves. Also, the testing of the program has proven showed that through use of 'try catch' statements it is ensured that there is little to no crashes, but this could improve very easily in the future to allow real input validation, for example if a user was to enter the wrong date format a popup would appear asking them to retry, and keep the form with all of the previous data. The actual validation on the KwikMedical prototype would be liaised further with the client before any actual validation would be added (client would specify what input is acceptable).

4.2.2 GPS (global positioning systems) technology

Currently, an ambulance is assigned based on the region of the patient, so that the hospital in that region will have an ambulance allocated. The current prototype does not deal with distance between patients and ambulances, but in the future the main goal of KwikMedical will be to equip ambulances with GPS and definitively find an ambulance which is closest to the area of emergency.

4.2.3 Distributed databases

To use decentralised databases would be another possibility, which would improve efficiency and allow each hospital to store data separately from headquarters database. This functionality would be required when a database becomes too large or too many clients are using it at one time causing bottlenecking.

5 References

Liu, x. - Data centred architectural styles (taken from Moodle, week 4).

Liu, x. - Aspect oriented software architecture (taken from Moodle, week 10).

Liu, x. – Remote invocation (taken from Moodle, week 6).

Liu, x. – Peer-Peer Systems (taken from Moodle, week 7).