2016

Software Architecture

COURSEWORK

AITKEN, JORDAN

Word Count: 2537

Table of Contents

1. Architecture Recommendations

There are several viable architectures that may be used to design the software for KwikMedical. The KwikMedical software is very reliant on database systems, and therefore an architecture that allows the most effective use of database systems are ideal. To reflect this, the two architecture style that have been chosen are **Aspect-Oriented** and **Repository**.

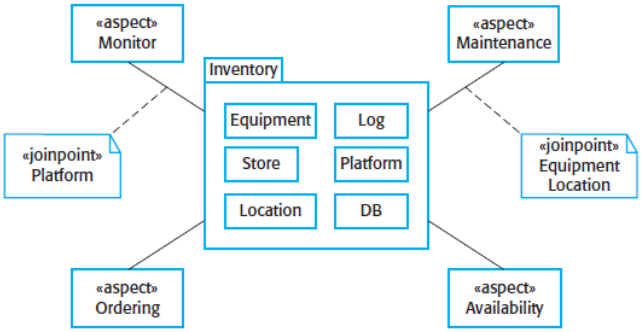
1.1 Aspect-Oriented

Some aspects of system implementation, such as logging, error handling, standards enforcement and feature variations are notoriously difficult to implement in a modular way. The result is that code is tangled across a system and leads to quality, productivity and maintenance problems. Aspect Oriented Software Development enables the clean modularization of these crosscutting concerns. (AJDT, n.d.)

1.1.1 What is the Aspect-Oriented Software Architecture Style?

Aspect-Oriented Software Architecture (AOSA) is an up-and-coming architecture that is gaining wide use. It addresses the problem of a piece of software having several interconnected parts, which can make it difficult to modify the code without breaking another part of it.

ASOA addresses this problem by using a new type of abstraction called an *aspect*. An aspect is used alongside other abstractions such as objects and methods.



(Liu, Aspect-Oriented Architecture Style, 2016)

1.1.2 Why Aspect-Oriented?

AOSA systems are primarily centred around a central data source. Other components can then extend from the central core. This means the program is easily extendable, and features can added in the future. Using an AOSA style means that a new feature can be built into the program as an extension or a plugin. This means that there are little to no difficulties adapting the code to the program.

1.1.3 Advantages

Wide Use.

AOSA is an architecture style that is gaining a lot of popularity. This means that it is widely used, and therefore a lot of developers are familiar with this architecture type.

Promotes Reusability.

AOSA affects code abstractly and does not interfere or upset other parts of the code. Aspects can be called and recalled multiple times. This prevents time being wasted writing code when existing code can simply be reused.

Simple; maintainable and readable.

Because each aspect is an extension to the central core. It is very easy to edit or change a single aspect without having a significant effect on other aspects. This makes it straightforward to identify issues and makes it easier to fix them.

Complements other architectural styles.

AOSA is an auxiliary architecture type, which means that it can be used to assist other architecture styles.

1.1.4 Disadvantages

Testing is difficult.

There is no standard way to test the Aspect-Oriented architecture style. There are several issues, for example:-

- Aspects must be specified so that tests can be derived
- Aspects must be able to be tested independently
- Aspect interference must be tested

Requires previous knowledge to read the code.

The use of aspects makes the program resemble a web, more than a linear style. Therefore knowledge of how aspects work is important in understanding how to read the code.

Difficult to read on paper

Due to the web-like structure, it can be overwhelming to tell exactly what is going in when the system is not in action.

1.2 Data-Centred - Repository

There are two categories of the Data-Centred architecture: **Blackboard** and **Repository**.

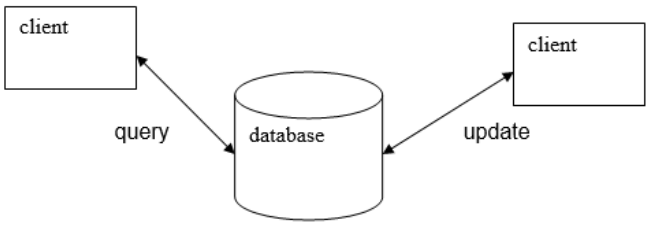
Both make use of a central data store. However, the difference that the Blackboard data store is active, while the Repository data store is passive.

1.2.1 What is the Repository Architecture Style?

The repository is a data-centred architecture style. It uses a passive repository that users can query and add data to.

There are two components used – The data store and the clients. Agents connect to the database via product protocol, for example JDBC.

The components connect using SQL queries to interact with the database.



(Liu, Data-Centred Architectures, 2016)

1.2.2 Why Repository?

KwikMedical uses a passive repository to store its patients and its details. Therefore it is a good idea to adopt the Repository style. It allows clients to query, update and insert data into the database from their own system.

1.2.3 Advantages

Data is secure and easy to backup.

Selection of a repository is possible. There are many reputable database-hosting programs available, such as WAMP, MySQL Workbench and Oracle Express. Security is one of the highest priority of a database system. Therefore there are many options available.

Can improve performance by locating on powerful machine.

A database is only as powerful as the machine it is being hosted by. This means that the database can be relocated to a more powerful machine if the current one is unable to cope with the traffic. It is a fairly straightforward process, and is a quick, albeit, expensive way to increase performance.

Easily readable.

The design of this architecture style is rather simple. There is a central database that clients can connect to send queries and updates to the database.

1.2.4 Disadvantages

Centralised data causes bottlenecks.

There is only one central data source that all clients are connected to. This can potentially cause bottlenecks if too many clients try to connect to the database at once. It can also put substantial load onto the database.

This can cause issues such as crashes and delays when working with the database.

System is vulnerable to failure.

If the central data source fails, then all connected clients will also fail as they cannot connect to the central data source.

Must maintain data integrity.

Data must be accurate to work with the data source. The data source will not accept any data that invalid. For example, it will not accept a String value to be inserted into an INT field.

2. Final Recommendation

The chosen Software Architecture style will be heterogeneous. This means that the chosen architecture style will be a combination of multiple styles. The chosen components of this heterogeneous style will be **Repository** and **Object-Oriented**.

2.1 Components

The components are **Repository** for the database and **Object-Oriented** for the client**.**

2.1.1 Repository

It is optimal to make use of a passive repository to store the data of the patients. A passive data-centred architecture style has been chosen because the data is only being used when a client makes a request, rather than the data always being active.

All hospitals will make use of the same patient records table, with the hospitals being clients and the records being the database.

2.1.2 Object-Oriented

While creating the system for the client to use. There are many object-oriented techniques that will be useful. For example, data encapsulation and polymorphism.

Encapsulation of data promotes reuse and hidden from the user. This makes the program flow effectively and produces a pleasant interface for the user – they don't need to be particularly savvy to interact with the program and it makes the system easy to use.

Polymorphism also promotes code reuse. For example, instead of manually creating 20 hospitals. Polymorphism can be used to create 20 Hospital objects using a hospital class. Each hospital can then have its own data.

2.2 Summarisation

- The central data source will be used store the patient records.
- The clients will be the hospitals, PDA and operator.
- Secure way to host information on the patients.
- Easy-to-use interface to update and query the database.

3. Design

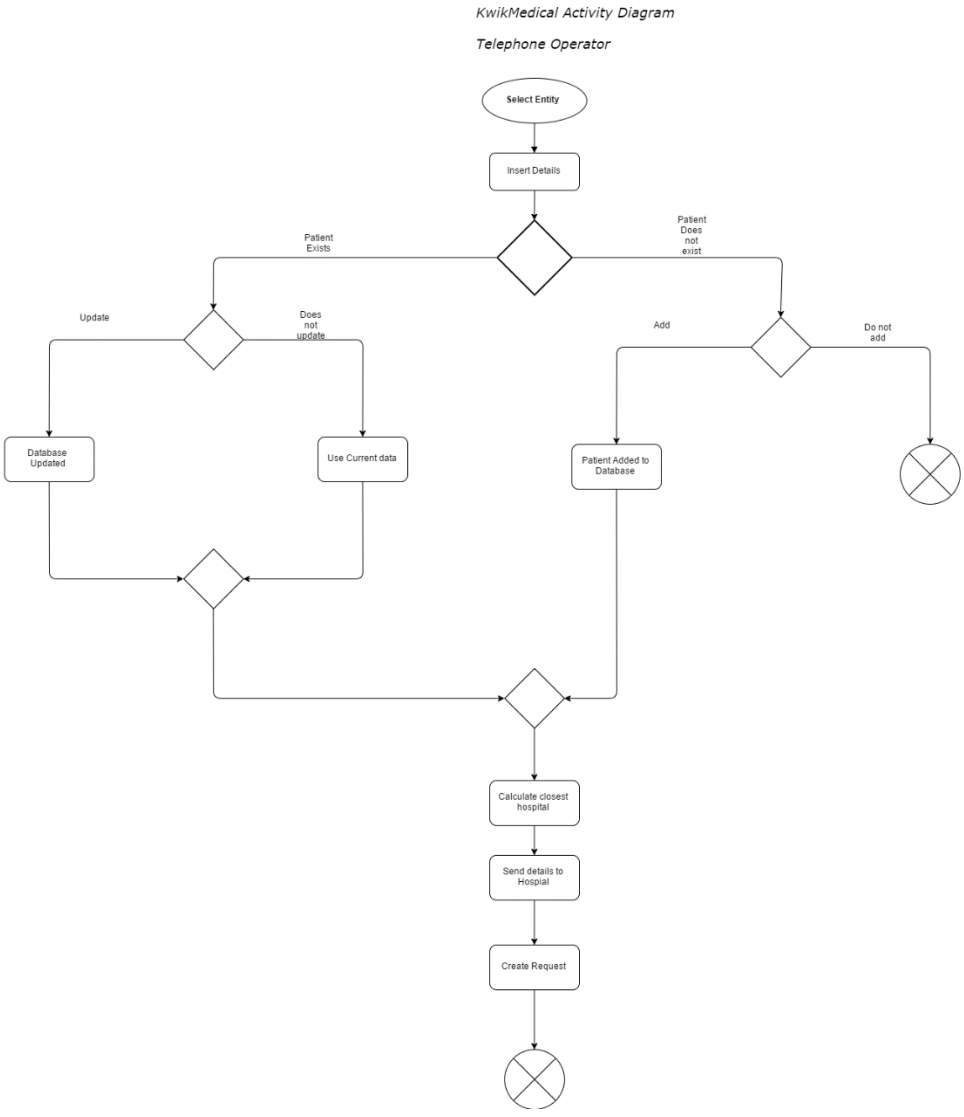
3.1 UML

UML (Unified Modelling Language) will be used to help plan out the project

3.1.1 Activity Diagram

Activity diagrams will be used to represent the flow of the program.

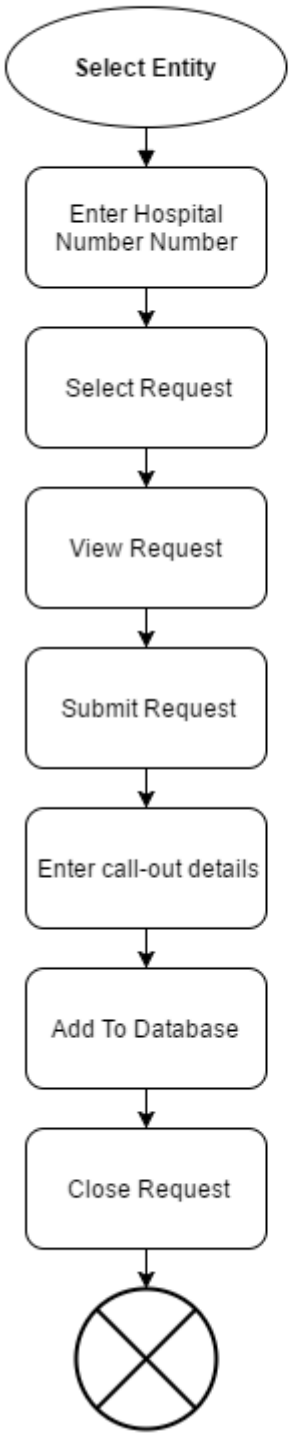
3.1.1.1 Operator



3.1.1.2 PDA

KwikMedical Activity Diagram

PDA



3.2 MoSCoW

The MoSCoW system is used to decide the significance of each specification using the following criteria, ranked most important to least important:

M: Must Have
S: Should Have
C: Could Have
W: Won't Have

ID	Classification	Description
1	Must Have	Receive input containing patient name, NHS Registration number, address and medical condition
2	Must Have	Use a passive repository that can be queried and used to insert data
3	Must Have	Verify existence of patient in the medical database
4	Must Have	Assign patient to hospital, and send records to that hospital.
5	Must Have	Hospital assigns an ambulance to the patient
6	Must Have	Send request to allocated ambulance containing details
7	Must Have	Ambulance can enter call-out details using their mobile device
8	Must Have	Comply with the Data Protection Act
9	Should Have	Update a patient's details with new information
10	Should Have	Add a patient to the database if they do not exist in it
11	Should Have	Close request once completed
12	Should Have	Work out closest hospital (extension of 5)
13	Should Have	Have a GUI (Graphical User Interface) to interact with the software
14	Should Have	Full input validation
15	Should Have	Database duplication prevention
16	Should Have	Have the ability to store multiple pieces of information about the patient (Eg. Multiple conditions)
17	Should Have	Link between the patient records and call out details
18	Should Have	Code should be well-commented
19	Could Have	Offer temporary treatment
20	Could Have	Ability to delete a patient
21	Won't Have	"Real" components such as an actual PDA or real hospital
22	Won't Have	Use GPS (Global Positioning System) to optimise vehicle assignment
23	Won't Have	User and Administrator Guides

This can be used to evaluate the program after implementation and testing.

3.3 Tools

The following tools will be used:

3.3.1 Language

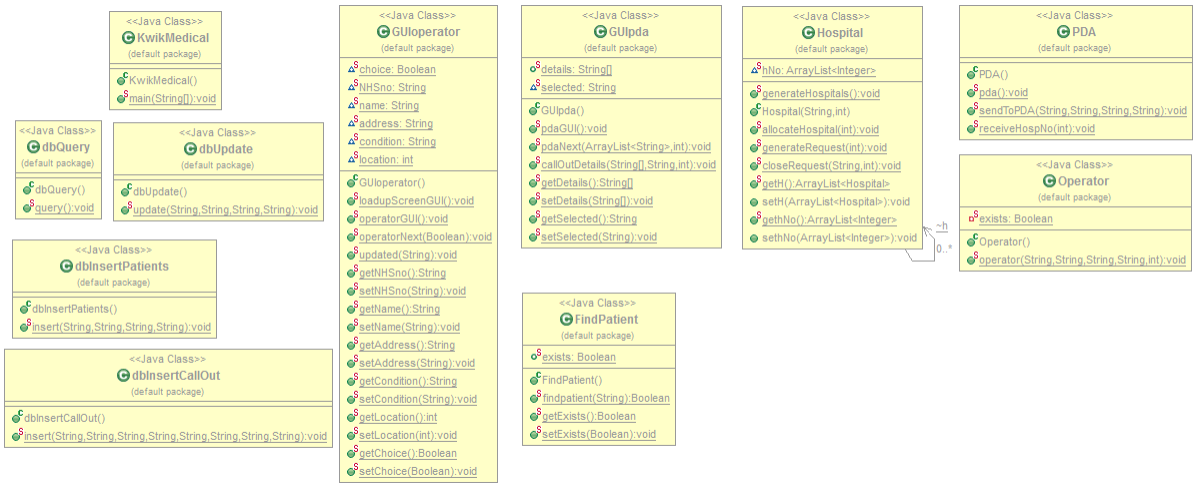
The language used will be Java, and will be developed using the Eclipse IDE.

JDBC (Java Database Connectivity) will be used for the API (Application Programming Interface).

3.3.2 Database

WAMPserver64 will used to host the patient records.

3.4 Class Diagram



3.5 Prototype Representation

- Entities will be represented using classes.
    - Operator
    - Hospitals
    - Ambulances
    - PDA/Smart Phone
- Requests will be simulated using text files.
- Closest hospital will be calculated using basic values to represent distance.

3.6 Databases

3.6.1 patientrecords

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	PatientNHSNo	int(10)			No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
2	PatientName	varchar(100)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
3	PatientAddress	varchar(200)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
4	PatientCondition	varchar(100)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values

3.6.2 calloutdetails

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	NHSNo	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
2	PatientName	varchar(50)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
3	pCondition	varchar(100)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
4	Date	varchar(10)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
5	Time	varchar(5)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
6	LocationOfAccident	varchar(200)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
7	ActionTaken	varchar(200)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
8	TimeSpent	varchar(4)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values
9	AmbulanceNo	varchar(4)	latin1_swedish_ci		No	None		Change Drop Primary Unique Index Spatial Fulltext Distinct values

4. Evaluation

4.1 MoSCoW

ID	Classification	Description	Complete?
1	Must Have	Receive input containing patient name, NHS Registration number, address and medical condition	✓
2	Must Have	Use a passive repository that can be queried and used to insert data	✓
3	Must Have	Verify existence of patient in the medical database	✓
4	Must Have	Assign patient to hospital, and send records to that hospital.	✓
5	Must Have	Hospital assigns an ambulance to the patient	✗
6	Must Have	Send request to allocated ambulance containing details	✓
7	Must Have	Ambulance can enter call-out details using their mobile device	✓
8	Must Have	Comply with the Data Protection Act	✓
9	Should Have	Update a patient's details with new information	✓
10	Should Have	Add a patient to the database if they do not exist in it	✓
11	Should Have	Close request once completed	✗
12	Should Have	Work out closest hospital (extension of 5)	✓
13	Should Have	Have a GUI (Graphical User Interface) to interact with the software	✓
14	Should Have	Full input validation	✗
15	Should Have	Database duplication prevention	✗
16	Should Have	Have the ability to store multiple pieces of information about the patient (Eg. Multiple conditions)	✗
17	Should Have	Link between the patient records and call out details	✗
18	Should Have	Code should be well-commented	✓
19	Could Have	Offer temporary treatment	✗
20	Could Have	Ability to delete a patient	✗
21	Won't Have	"Real" components such as an actual PDA or real hospital	✗
22	Won't Have	Use GPS (Global Positioning System) to optimise vehicle assignment	✗
23	Won't Have	User and Administrator Guides	✗

4.2 Future Development

Closure of completed or deleted requests.

Once a request has been responded to, or removed, the program should close the request, but still keep a record of it.

Work has been started on implementing this feature, and it is able to identify a request that must be closed. However, due to time constraints, the feature is incomplete as it does not actually close the request or prevent the program from opening the requests

Validation of input.

At the moment, the program does not handle invalid input very well. This was not implemented due to time constraints.

The program deals with best-case scenario user input. I.e. User enters all fields correctly.

However, the program does not currently deal with incorrect field entries. For example, leaving fields blank, entering hospital numbers that does not exist.

Offer temporary treatment suggestion.

This could be implemented by having a database containing medical conditions, and upon having a match, the program can suggest the treatment.

For example, using cold water on a burn.

Allow assignment of ambulances.

Due to time constraints, each hospital only has one ambulance that is assigned to the patient. In the future, a new table of ambulances should be created with a 1-to-many relationship (One hospital has many ambulances)

Ability for a patient to have multiple conditions at once.

A current issue with the program is that a patient can only have one condition at a time. This causes a problem if a patient breaks their leg while they are already registered with faulty liver, for example. This can be implemented by having a table of conditions with a 1-to-many relationship (One patient can have many conditions).

Ability to delete a patient.

An ability to delete a patient would be possible to enable in the future. There was some ambiguity if it was appropriate for an operator to be able to remove patients from the database.

However, it is a fairly straightforward process to enable the ability to delete patients.

Implementation of real components.

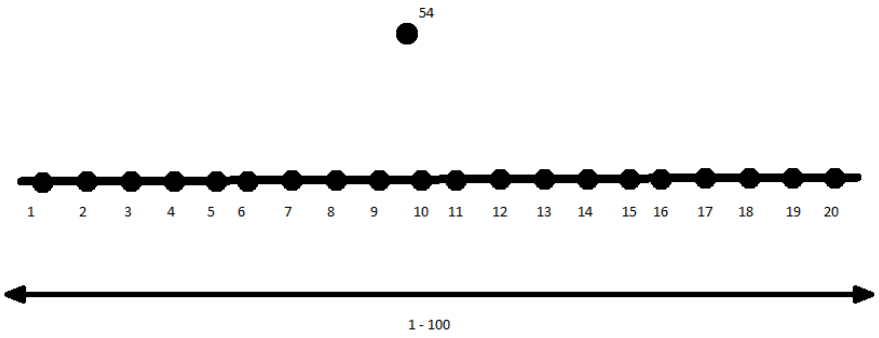
The program is currently a prototype, and therefore does not use real components such as ambulances, hospitals and mobile devices.

However, these can be added in future development.

Use of GPS to optimise hospital assignment.

GPS can be used to create a real hospital assigning system.

The current system is in a way that the each hospital has a value, ranging from 1-100 at intervals of 5.



The user enters a value to represent their location, and the program assigns the hospital that is closest to the patient.

In the future, this can be replaced by using the patient's address and the address of the hospitals using a system such as Google's API.

User and Administrator guides.

Guides for the user and administrator will be written in the future, but at the moment, these are out of scope for the prototype.

5. References

AJDT. (n.d.). Retrieved from Eclipse: http://www.eclipse.org/ajdt/

Liu, X. (2016). *Aspect-Oriented Architecture Style.* Retrieved from Moodle.

Liu, X. (2016). *Data-Centred Architectures.* Retrieved from Moodle.