Interactive Access to Museum Information using Visitor's own Mobile Devices

Lea Tracy Savage

Submitted in partial fulfilment of
the requirements of Napier University
for the Degree of
Software Engineering

School of Computing

December 2007

Authorship declaration

I, Lea Tracy Savage, confirm that this dissertation and the work presented in it are my own achievement.

1. Where I have consulted the published work of others this is always clearly attributed;
2. Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own
3. work;
4. I have acknowledged all main sources of help;
5. If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;
6. I have read and understand the penalties associated with Academic Misconduct.
7. I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Date:

Matriculation no:

Data Protection declaration

Under the 1998 Data Protection Act we cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

Development within the mobile device industry in previous years has enabled the use of portable, handheld access. The devices used for such access are proficient in communicating using Wireless and Bluetooth networks, able to load colourful imagery and sound files and, advantageously, enable the user to take more control of the content they receive.

The project is based on the conclusions outlined in Celia Curnow's Master of Science (MSc) project, where a web based prototype was developed for visitors to a pottery exhibition within The Royal Museum Edinburgh, using Personal Digital Assistants (PDAs). This project has taken the feedback results and modified the idea to improve the overall effectiveness and use of an interactive guide within a museum environment.

Concurrent software will be developed as inspired by Hoare's communicating sequential processes notation. Multiple mobile devices will be capable of interacting with different artefacts using wireless communication and the concept of mobile processes. The concurrent software will prevent the use of museum based equipment and there will be no costs incurred by the user.

JCSP will be used in conjunction with Java and its new scripting language, Groovy, with a support database created in Apache Derby. Grails, a new dynamic web language, will be used to create a management system to provide museum staff with an attractive interface for updating items stored in the support database.

Contents

Use menu bar Insert; Reference; Index and Tables; Table of Contents to level 2 only

List of Listings

List of Figures

Acknowledgements

This project relies heavily on the Java Communicating Sequential Processes (JCSP) framework so I would like to thank Peter Welch at the University of Kent for its development and the use of his site and presentations to gain a better understanding of the technology. The past few years has established a useful contributor to JCSP, Kevin Chalmers. He has spent a considerable amount of effort improving the initial concepts to create JCSPMobile which is the main backbone to this project and therefore would have been impossible without. During development I had regular contact with Kevin which without, the project would not have been as successful.

I would like to thank Sun Microsystems for the development of Java and their useful, well maintained documentation of the platform and its various APIs. I would also like to thank Sun Microsystems for providing the inspiration for the development of Groovy. Groovy which was developed by the Codehaus team is a useful and intelligent scripting language for Java. Its dynamic concepts helped to integrate the different components of the system including the database, conversion of objects and Grails with ease. I would also like to thank Graeme Rocher and his team for the development of Grails which uses the concept of Groovy to create an object orientated web application that can be easily integrated with other Java based applications.

Without the development of Derby by the Apache Software Foundation, I would have been unable to successfully store the images, sound and textual descriptions of the museum's artefacts.

The use of FreeTTS and its demo examples enabled me to create sound files for the application that otherwise would have been difficult to implement.

Finally I would like to thank the School of Computing teaching staff at Napier University for their support highlighting, in particular, Professor Jon Kerridge for his regular support and encouragement. His idea and underlying methodology and research provided the starting point and main theme for the project.

I would also like to thank my proof reader and boyfriend, Bryan Fraser for his support and patience whilst I wrote and developed this dissertation.

1 Introduction

The desktop computer phenomenon has inspired the mobile industry to research the ability to develop small powerful handheld devices. The processing power of computers has increased considerably and therefore created an opportunity to research and experiment with alternative technologies. What was once restricted to desktop computers, has now reached the ability to be mobile, inspiring new possibilities to be developed. Therefore this project takes advantage of these new emerging developments within the scenario of a museum environment. Memory capacity is an area with difficulty due to the restrictiveness of 'handheld'. Therefore development techniques need to be efficient yet small to ensure effective performance within suitable time constraints.

1.1 Motivation

Curnow (2006) investigated the use of technology within a museum environment and surveyed the visitors' experience, expectations and the long term benefit to the museum. This project extends this research to create new interactive opportunities for the visitor without increasing the workload for the museum and preventing the fun factor being lost when combining tourism with new technology.

1.2 Background

In the past few months Google, Yahoo! and Microsoft (PDA Essentials and GPS Advisor, 2007) have developed and launched websites dedicated for mobile viewing. Experts in their field, in particular search engines, these competitors have noticed the popularity and interest in this ever expanding sector. Only recently have mobile devices been considered an alternative to the desktop computer for innovative ideas and development.

Improved processing speeds, increased screen sizes and pixel quality, ensure these devices are accessible to the consumer for prolonged activity. They are now less regarded

as a phone and more as a portable, handheld computer that can act as a MP3 player, video player, games console, electronic diary, camera or/and a web browser and let us not forget SMS which enables the phenomenon of text messaging.

This influx of technology has enabled developers to become portable with their applications and their ideas. With the development of Java Communicating Sequential Process (JCSP) Mobile package by Chalmers and Kerridge (2006), the ability to download parallel processes to a PDA became achievable and therefore this project will take advantage of this technology to create an interactive guide within a museum environment.

One thing not mentioned, which is a limitation, is memory capacity. To remain compact, mobile devices are restricted in memory allocation so development needs to be small and precise. Advantageously CSP/JCSP relies heavily on a central server so the mobile device optimally downloads the necessary code.

Mobile devices have evolved during the past decade to enable users to text, email and manage busy schedules but one of the major demands by users is the increase of battery life (Telecommunications Industry News, 2005). Lim (2006) wrote an article on tips to conserve battery life and states:

> *"It may be fun playing games or browsing the Web at the bus stop, but your battery will have run out by the time you get to work"*

Therefore accessing the web using a mobile device can be a strain on the battery and the alternative of downloading small programs is a suitable option. Visitors will not want the mobile device's battery to be consumed with the museum application as they may intend to use the facility once they have finished their visit. The programs developed using JCSP can vary from 1 to 10 KB which is considerably less to the 15MB daily restriction that comes with most mobile phone service providers (Vodafone, 2007).

1.3 Aim

The project aims to provide the visitor with the option of using their mobile device when visiting the museum. Adding this functionality will provide visitors with extra learning opportunities and may improve their experience. The technology will be versatile and users will be comfortable with its operation due to it being their own device. Such

devices will include Personal Digital Assistants (PDA), smart phones and cellular (mobile) phones.

1.4 Objectives

This project contains a variety of different stages, each adapting different technologies to combine the following objectives, eventually creating the prototype that will meet the above overall aim.

- The project intends to use Mobile Devices to interact within a museum environment. The first stage is to create a program that downloads effectively to a Mobile Device without being disruptive or unfriendly to the visitor.
- Once the basic interactions are working, building on this to include images that relate to the current item that the visitor is currently viewing.
- Mobile Devices are compact and therefore images will be small and, as a consequence, lack detail. Text will be short and precise so adding sound can represent a detailed description of the item and will add an additional feature to improve the interaction for the user.
- Databases are useful for storing data of a similar nature and as the information being used by the program will include text, images and sound, it seems like a suitable storing device for the project. As mentioned the database will store details of the items within the museum and when a visitor is near a particular item, the details of that item are retrieved.
- A web based interface will enable the museum staff to update, when required, specific details of stored items from within the Database (this objective has been added during the course of the project).
- Investigate Bluetooth as a potential communications medium for the museum environment.

The above objectives relate to the program that will run on the mobile device, however there needs to be a mechanism in place to detect a mobile device. Bluetooth and wireless are two communication protocols and it is uncommon for mobile devices not to include these features already built in. Therefore using these protocols, mobile devices can be detected and the program can be downloaded using IP addresses.

1.5 Structure

Chapter two will discuss the technology currently available within a museum environment and highlight their advantages and disadvantages. Mobile devices will then be described evaluating their current functionality and use within a museum environment. The final section of this chapter will mention the current software available for developing such a proposed system.

Chapter three will build on the key evaluations from Chapter two and use design techniques such as PACT Analysis and UML to complete the full design of the system. Chapter four will then take the user requirements listed in Chapter three and implement the system using an incremental approach. Chapter five is an extension to the implementation process which will discuss the development of the Grails application for use by the Museum staff.

Chapter six will take both systems and test their current functionality to determine whether or not they meet the user requirements and their behaviours are as expected. Chapter seven will elaborate on testing focusing on the User's perspective and requirements discovered during the PACT analysis.

Chapter eight will discuss the possibilities of integrating Bluetooth within the system and evaluate its appropriateness in a museum environment. Lastly Chapter nine will evaluate the finished prototype against the objectives outlined in Section 1.4 and discuss outstanding and future work.

Throughout the report Listings will be displayed showing various code examples that are relevant to the section. For ease of reference,, numbers have been added to each listing to represent individual lines. When lines are discussed within the text the corresponding number will be surrounded by {}, for example {34 – 35} means lines 34 to 35 inclusive.

2 Literature and Technology Review

This chapter discusses the current opportunities available within a museum environment and how these can be adapted and improved to achieve a suitable interactive environment. The project would not be successful without an understanding of what technology is currently attainable and how these can be combined to meet the overall aim of developing an interactive application that will run on a mobile device. The latter part of the chapter discusses the current technology available and its expected use.

2.1 Museum Environment

This project is a natural progression from the conclusions discovered by Curnow's (2006) MSc Dissertation. Curnow's main objective was to research and observe the benefits of interactive technology within a museum environment, in particular a pottery exhibition.

There are many interactive applications available in museums throughout the country but not all of these have replaced traditional human tour guides and are unlikely to do so. As part of this section, current technology in museums will be described and evaluated. Concentrating, in particular, where they are not successful and why and how they can be modified.

Appreciation of a museum visit comes with age and with it the ability to spend endless hours reading information boards about ancient artefacts that the younger generation cannot appreciate as they do not have the patience. When designing an application a key factor to consider is the target audience. Curnow used a website to implement the guide; however on a mobile device this would incur a cost for individuals. It would be difficult to interact with the user without connecting their device to the internet before they could close the connection, this process would incur costs before the guide had even started and result in an expensive, distasteful experience for the visitor. Therefore this design needs

to interact with the user without the costs and ideally, minimising the affect of their device's performance. Curnow's prototype received negative feedback about the use of bookmarks (2006, page 3) and therefore highlights the need to prevent information remaining stored on the mobile device once they have left the museum.

Karpf (2002) wrote an article detailing a visit to London's Science Museum focusing on interactive devices and its relevance with learning within this environment. In terms of technology, the article is negative but the reasons are justified and therefore need to be considered within this project's design.

> *"In fact a number of the buttons and levers are broken, and the ones that aren't encourage a hit-and-run approach. Kids press or push and move on, without waiting for the result - a bit like ringing doorbells and running away."*

This design should, therefore, focus on the technology and the user and not the museum and the technology, as a result only visitors that have the knowledge in using the technology will have the ability to use it. Other visitors will continue their visit without being distracted or encouraged to use a device that they would otherwise have been unaware of.

Petrelli and Not (2004, Section 3.2) studied the trends in museum visitors in particular they wrote the following identifying that the majority of people are returning visitors:

> *"…the number of non-first-time visitors, accounting for 68% of the sample…Returning visitors came to see specific objects and stayed in the museum longer…skipping an object may indicate lack of interest, but this may not be the case for frequent visitors who have seen the object before."*

Their survey, completed in 3 different museums, identified this trend and therefore should be considered when developing an application to interact with a user as it cannot interrogate the visitor to use the application. As part of the design, a feature to exit the application must be available to encourage visitors to continue using the application when they revisit to observe new items instead of switching off their mobile device before entering the building.

The high percentage of returning users may also suggest the requirement for more information. Museums usually lack dynamic information and therefore a returning user

will struggle to find out more information of an area that is of interest unless it is a special exhibition.

Another disheartening trend identified in the same survey was:

> *"Only 7% reported to like using technological devices as museum guides. Most people liked visits guided by a member of the museum staff (53%)…but it also suggests that listening to a human guide is still the easiest way to get information. Finally, the general dislike of technology suggests that some visitors may never explicitly interact with the system."*

Although this is one survey, Curnow had similar results (2006, Section 6.2.1) where the users had no prior knowledge of using a PDA and therefore without human support, would have struggled with the technology. Therefore using the visitor's own mobile device with this design is vital in reducing the high percentage of unfamiliarity. The interaction will be optional and available for people interested in technology. Pretelli and Not suggest that 53% of visitors liked a human guide and listening to information, adding sound to this application will hopefully enhance the guide and reduce the differences between a human and mobile device guide.

2.2 Mobile Devices

The mobile device has evolved since the 1980s when they were mainly regarded as an efficient method of emergency contact and a luxury accessory. Recent developments and innovations have surpassed expectations making it an affordable and essential addition to our everyday lives.

The desktop computer has become a popular electronic device that society has wholeheartedly embraced. The mobile industry has observed both technologies and, with the development of mobility made possible by laptop and notebook computers, would like to achieve similar success. Mobility improves the interactivity options available for technology and additionally mobile devices encourage new portable possibilities due to their compactness. As a result they have become a suitable alternative to the desktop computer but their performance is sacrificed due to the demand of a regular power supply and countless accessories.

There are three different categories of mobile device currently on the market:

- *Limited Data Mobile Device* – The original portable mobile device that offers a simple text based screen with functions such as Short Message Service (SMS) i.e. text messaging, access to the internet using the Wireless Application Protocol (WAP) and standard telephone capabilities.
- *Basic Data Mobile Device* – The mobile device evolved from the text based screen to offer an icon based menu structure using a scroll or joystick navigational tool. Additional features now include email and a contact library. One popular product in this category is the BlackBerry, a wireless mobile device created in 1999 by Research In Motion (RIM), and Smartphone which was launched in 1992 by IBM.
- *Enhanced Data Mobile Device* – An extension to the Basic Data Mobile Device but offers a stylus based navigation plus popular Windows Office applications including Word, Excel and PowerPoint. A well known product in this category is the Pocket PC.

Within these categories was the discussion of wireless as a communication protocol. Current mobile phones offer different mechanisms to wirelessly communicate including:

- Global System for Mobile Communications (GSM) is a wireless technology that enables mobile phones to internationally roam using unique bandwidths for communication ensuring they remain contactable when travelling abroad.
- General Packet Radio Service (GPRS) is available as part of GSM to enable enhanced features including connecting to the internet and the retrieval of emails. The latest development is the Enhanced Data rates for GSM Evolution (EGDE) which is capable of up to three times the data capacity of GPRS.
- Wireless Fidelity (WiFi) is a wireless communication protocol for local area networks (LANs). Mobile phones can connect to a Wireless Access Point and then be treated as a node on the local network which is then capable of downloading data and connecting to the internet.
- Bluetooth is another wireless technology but used within a small range and normally peer to peer. Inspired by infrared technology where both devices needed to have a direct link, Bluetooth does not have this requirement and is therefore more versatile.

The above communication protocols have improved the facilities available for mobile phones and helped them to be more alike with the desktop computer. The other benefit of these protocols is the lack of cables and installation requirements. Mobile phones can

naturally detect the different communication technologies available within range without the user's participation.

2.3 Concurrency/Parallelism

A concurrent application is commonly defined as a program with more than one active process running at the same time. The process can be on a single machine or several processes on multiple machines communicating together. Using the notion of preparing a meal (Bustard, Elder & Welsh, 1988) Figure 1 shows the main activities for preparing a meal in a concurrent system.

```
Open refrigerator
IF refrigerator is empty
THEN eat at restaurant
ELSE
  BEGIN
        Prepare hors-d'oeuvre
        Prepare entrée
        Prepare dessert
        Eat at home
  END
```

Figure 1: Bustard's sequential process example

The key activities of interest are the *prepare* statements. Within a single process the activities are completed sequentially but by adding another process the result is more than one of the activities being completed simultaneously as shown by Bustard below in Figure 2:

```
John:
  Open refrigerator
  IF refrigerator is empty
  THEN eat at restaurant
  ELSE
    BEGIN
        Prepare hors-d'oeuvre
        Prepare dessert
        Eat at home
    END
```

```
Catherine:

  IF refrigerator is empty
  THEN eat at restaurant
  ELSE
    BEGIN
        Prepare entrée

        Eat at home
    END
```

Figure 2: Bustard's example with two processes

Adding further processes improves the overall computational time of the program. The above example highlights the effectiveness of concurrency but there are many known issues that can result in problems.

Continuing with the food theme, the Dining Philosophers problem is a common method of describing concurrency problems and was used by Hoare (1985) during the development of Communicating Sequential Processes (CSP) based on a problem by Dijkstra (1968).

Five philosophers are sitting at a circular table, shown in Figure 3, and are capable of doing 2 things; eating or thinking. There is a large bowl of rice in the middle of the table and in between each philosopher is a chopstick. Each philosopher will have one chopstick to their left and one to their right and can only use these chopsticks to eat the rice.

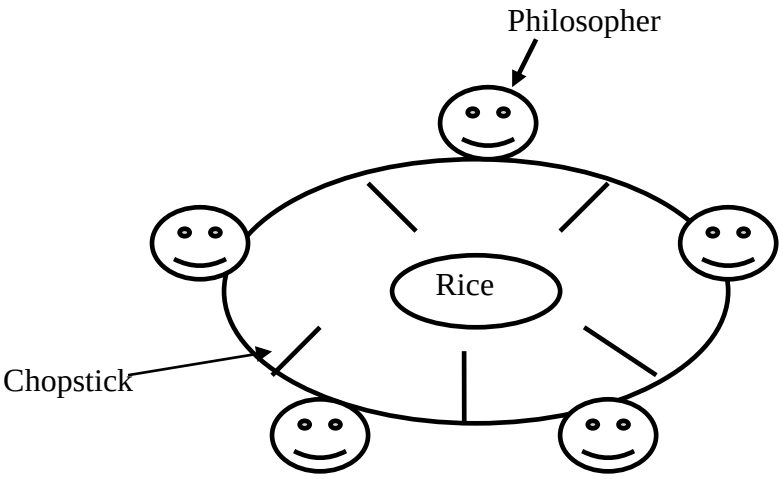


Figure 3: The Philosopher example by Hoare

The philosophers cannot communicate with each other. Imagine each philosopher selects their left hand chopstick and then tries to obtain the right hand chopstick. Their neighbour will already have it and therefore deadlock occurs. Deadlock is the most common problem in concurrent programs and exists when neither process can complete their task and therefore continues in an infinite wait loop. Starvation happens when a process is ready but never has a chance of completing their task within the expected timeframe because another process is using the required resources. In the dining philosopher problem one philosopher may always miss retrieving the 2 chopsticks before another philosopher starts eating again.

To prevent deadlock a time delay may be implemented so that a philosopher waits 2 minutes before trying to obtain both chopsticks. However starvation may occur if the philosopher always misses an opportunity during the 2 minute delay. The time delay also

causes another problem called Livelock. Livelock is caused when each philosopher picks up their left chopstick at the same time, notices the right is unavailable and therefore delays for 2 minutes. Each philosopher will then try again 2 minutes later and it will result in the same scenario and therefore an infinite loop.

2.3.1 Programming Techniques

There has been significant development in concurrent programming techniques to resolve the above issues that relate to shared resources. Whiddett (1987) describes some useful paradigms to overcome the difficulties associated with concurrent programming; monitor, message and operation orientated.

Dijkstra (1988) created the concept of a semaphore to enable resources to be released and acquired by processes using *wait* and *signal*[1] for notification. Semaphores are versatile and therefore complicated and difficult to maintain if the system is prone to updates or restructuring. This vulnerability resulted in the concept of a Monitor. Monitors are passive objects that control the mutual exclusion of their data from processes within the application. This method prevents two processes simultaneously updating the shared resource causing corruption or out of sync data.

Message based interaction is similar to the structure of general Input/Output where there can be two types of action; send and receive. Message based interaction implements the same structure where processes send and receive access to shared resources via a general process which manages the requests.

Operation orientated is the concept of connected processes using the techniques taken from the monitor and message based systems. Several systems including Ada and the method 'distributed processes' took the experience of the previous paradigms to enable distributed systems (more than one processor) and efficient synchronisation and remote procedure calls.

2.4 Communicating Sequential Processes

Hoare (1985) originally presented a theoretical notation for developing concurrent programs using abstract descriptions of processes; an object or entity that exists independently and can communicate using events or actions; forms of message passing between processes. The algebraic notation and graphical representation is similar to

[1] Dijkstra used the Dutch notation P and V but English translation is *wait* and *signal* respectively

Figure 4 using the chocolate vending machine example as described by Hoare where a coin is consumed.

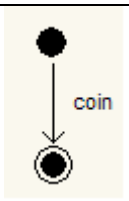
(coin →STOP $_{\alpha VMS}$°)	

Figure 4: Hoare's chocolate vending machine example in textual and graphical form

Hoare elaborates this example to take in the notion of concurrency. Two processes; vending machine (VM) and customer (CUST) have the following events:

VM = {coin, choc, clink, clunk, toffee}

CUST = {coin, choc, curse, toffee}

Figure 5: Chocolate Vending Machine example with two processes

The customer would like to have a toffee bar but the vending machine is out of stock, therefore the customer will 'curse' when it receives the chocolate bar. The clink and clunk actions in the vending machine are the noise when a coin is inserted and the disposal of a chocolate bar respectively. Combining the two processes together results in the notation in Figure 6.

(VM||CUST) = μX.(coin → (clink → curse → choc → clunk → X
| curse → clink → choc → clunk → X))

Figure 6: Chocolate Vending Machine example with concurrency by Hoare

What Hoare focuses on representing in this example is the fact the clink or curse could happen first or simultaneously. He also points out that the notation abstracts from the customer's emotions and therefore reality as it does not represent the fact that they wanted a toffee bar from the vending machine.

Hoare's CSP notations, an example of a message based system, influenced the development of the US Ministry of Defence language Ada. Ada is a Real Time

° α_{VMS} identifies the process Vending Machine

programming language used in aircrafts and an example of one of the first object orientated systems. Occam, another inspiration from CSP, was a parallel processing language developed by Bristol based company INMOS in 1983. Occam's principles have been used to develop the Java based libraries (JCSP) (Welch & Brown, 2007).

2.4.1 Java Communication Sequential Processes

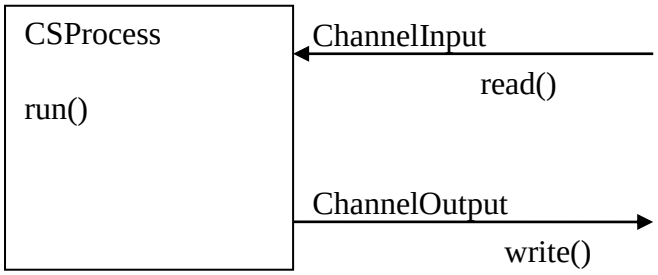


Figure 7: JCSP structure showing a Process with two channels

JCSP is a Java implementation of CSP developed by Peter Welch and Paul Austin at the University of Kent. JCSP removes the complications of multithreaded development in Java applications and avoids the common pitfalls of deadlock, starvation and livelock. JCSP uses encapsulated processes that communicate using non-buffered channels. When a process writes on its channel another process should be ready to receive the communication. A similar analogy is an independent serial program with input/output connections.

Figure 7 shows a structure of a process in JCSP. Each process implements the main interface `CSProcess` and must contain the method run. Processes can have a number of `ChannelInput` and `ChannelOutput` interfaces which consist of the method read or write respectively. If the process calls the read method of the ChannelInput then it will wait until something has been received. Similarly sending data on the ChannelOutput method halts the process until the data has been accepted by another process.

The common problems mentioned previously are resolved by CSP because processes are responsible for their own events and another process cannot interrogate a process without using the communication channels which halt executions until the receiving process is ready.

In JCSP every write process must have a corresponding read process or the write process will remain blocked. Listing 1 is an example of a basic write process that increments 1 to

100 {5, 7} and writes the result of `i` to its ChannelOutput `outChannel`, written in Groovy (Codehaus Foundation, 2006a).

```
1   class Producer implements CSProcess {
2     def ChannelOutput outChannel
3     void run() {
4       def i = 0
5       while ( i < 100 ) {
6         outChannel.write (i)
7         i++
8       }
9     }
10  }
```

Listing 1: Example of the Producer process

Listing 2 is the corresponding read process that receives the input {16} and outputs the result to the console {17}.

```
11  class Consumer implements CSProcess {
12    def ChannelInput inChannel
13    void run() {
14      def i = 0
15      while ( i < 100 ) {
16        i = inChannel.read()
17        println "the input was : ${i}"
18      }
19      println "Finished"
20    }
21  }
```

Listing 2: Example of the Consumer process

As mentioned previously the processes are independent and unaware of other processes, therefore another class, i.e. main, needs to be created to connect the two processes together as shown in Listing 3.

```
22  One2OneChannel connect = Channel.createOne2One()
23  def processList = [ new Producer ( outChannel: connect.out() ),
24                      new Consumer ( inChannel: connect.in() )
25                    ]
26  new PAR (processList).run()
```

Listing 3: The process that connects the Consumer and Producer processes with channel 'connect'

Within this main class is the mechanism of a channel {22} which creates the communication link between the two processes {23, 24}. The last stage {26} is to start all of the processes so they are working concurrently.

Using relational notation there are 4 different channels available as shown in Figure 8. As the names suggest the `One2OneChannel` is a singular connection between two processes and `Any`, depending on where, provides multiple writers/readers that compete for access to the channel. If there are multiple writers then only one of those writers can

use the channel to communicate with the reader at any given time. To maintain mutual exclusion of the resource, i.e. the reader, a `ChannelDataSource` is created as part of the channel. This acts as a monitor to manage the access to the resource.

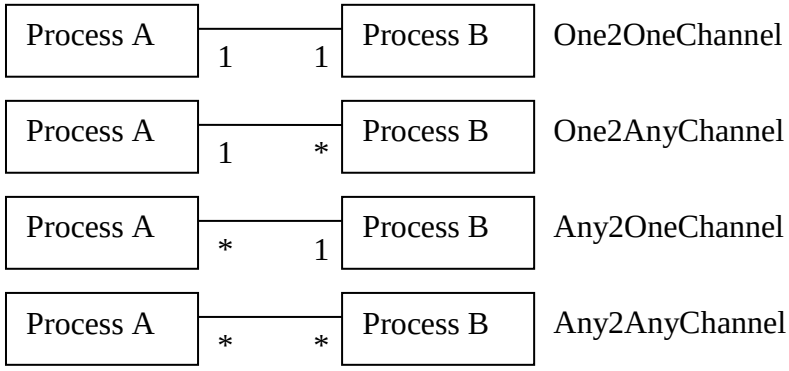


Figure 8: Examples of the different channels available in JCSP

Inspiration from Occam, JCSP has implemented five different Guard classes to manage the control of simultaneous events from different channels.

- ***AltingChannelInput(Int|Accept)*** – when you want to manage channels sending messages to limit the time they are pending the above three guards can be used.
- ***CSTimer*** – Is managed by a countdown mechanism so the process is alerted when the time has expired.
- ***Skip*** – This guard is always ready.

The notation of an Alternative combines the above guards into an array list and monitors when a guard's status is true or in other words, requires attention. The following listing taken from a presentation by Welch (2002a) is simple example of setting up an Alternative. There are 2 guards {28} with integer values 0 and 1. The Alternative {27} is set up to monitor these 2 guards. Within the Switch statement {30} the alternative has 3 options of how to manage the different requests.

- ***alt.select()*** – waits until more than one guard is requiring attention and makes an arbitrary choice.
- ***alt.priSelect()*** – similar to above but chooses the smallest index first.
- ***alt.fairSelect()*** – similar to the others but ensures all guards have equal precedence.

When a guard is ready its index number is matched with the corresponding case statement.

```
27  final Alternative alt = new Alternative (new Guard [] {event, in}; );
28  final int EVENT = 0, IN= 1;
29  while (true){
30    switch(alt.priSelect()){
31      case EVENT:
32        event.read();
33        event.read();
34        break;
35      case IN:
36        out.write(in.read());
37        break;
38      }
39  }
```

Listing 4: Example of Guards and Alternatives in a process

2.4.2 JCSPNet

JCSP provides a successful mechanism for developing parallel programs using a single Java Virtual Machine (JVM). Using the same notation however, JCSPNet (Welch, Aldous & Foster, 2002b) has extended this notation to enable several processes to work in parallel using a shared network link and therefore multiple JVMs.

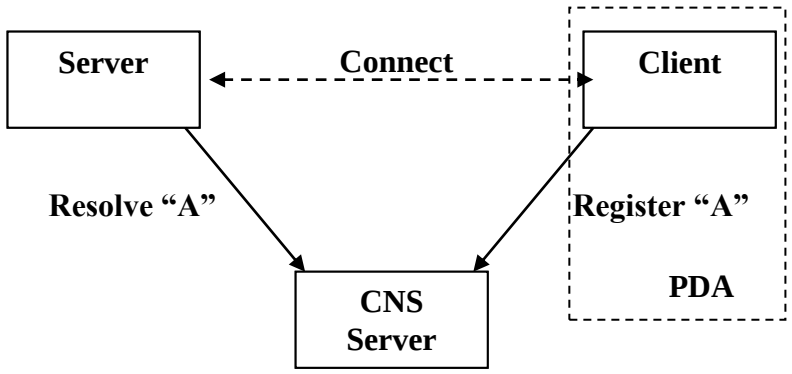


Figure 9: Establishing a connection between a Client-Server system using JCSPNet over a network

A `Channel Name Server` (CNS) on the network enables other processes to connect without knowing the locality of others on the network. Figure 9 is a diagrammatic representation of the framework. Processes register their read channel to the `CNS` as shown by Figure 9 where the `Client` registers "A". The `Server` then tries to resolve "A" via the `CNS` so a connection can be created between the `Client` and `Server` called "Connect".

JCSPNet therefore enables the dynamic construction of a network but still maintains the same workflow as JCSP with only one comparative difference; on a single JVM, objects are referenced whereas on a network the object is copied and serialised for efficient transfer over the network.

2.4.3 JCSPMobile

JCSPNet provided the mechanism of Dynamic Class Loading to enable processes to be downloaded to separate processors with the creation of dynamic channels to create the concept of "mobile". However documentation for this concept was sparse until the package JCSPMobile (Chalmers et al, 2005) was created. Using the concepts already available from JCSPNet, JCSPMobile refined, and documented, the ability to create mobile processes and channels whilst maintaining the existing features of JCSPNet.

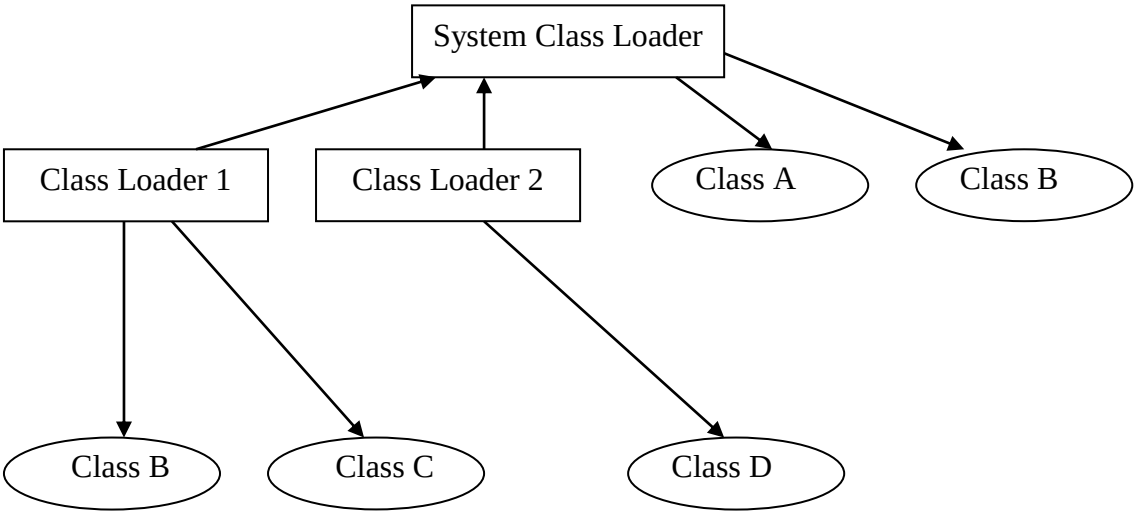


Figure 10: Class Loading in Java (Chalmers, 2006)

Chalmers et al describes Dynamic Class Loading using the diagram shown in Figure 10. Dynamic class loading enables the system to load classes to the runtime environment whenever required. There can be several class loaders within one system using a hierarchical tree structure. Lower nodes such as Class Loader 1 can call its parent, System Class Loader, to instantiate a required class but the parent tree cannot call the lower class loader for instantiation. In Figure 10 Class Loader 2 can get instantiations of Classes D, A and B but the System Class Loader can only get Classes A and B.

Chalmers et al have taken this functionality and applied it to a class `Mobile` in the JCSPMobile package and combined it with a refined version of the JCSPNet `FilteredChannel` object. This functionality enables details of the sending process to be attached before transmission and permits the mechanism to obtain any classes that are required to run the passed process.

JCSPMobile requires the client to have one process running in order for the above mechanism to work. The `MobileProcessClient` is responsible for creating the

connection to the CNS and therefore the ability to communicate with the server, Listing 5 is an example of the main steps of the `MobileProcessClient`.

```
40  public class MobileProcessClient {
41    String CNS_IP = Ask.string("Enter IP address of CNS: ");
42    if (CNS_IP == null) {
43      Mobile.init(Node.getInstance().init(new TCPIPNodeFactory()));
44    }else {
45      Mobile.init(Node.getInstance().init(new TCPIPNodeFactory(CNS_IP)));
46    }
47    String processService = "A";
48    NetChannelLocation serverLoc = CNS.resolve(processService);
49    NetChannelOutput toServer = NetChannelEnd.createOne2Net(serverLoc);
50    processReceive =  Mobile.createNet2One();
51    toServer.write(processReceive.getChannelLocation());
52    MobileProcess theProcess = (MobileProcess)processReceive.read();
53    new ProcessManager(theProcess).run();
54  }
```

Listing 5: MobileProcessClient code example (Chalmers et al, 2005)

Firstly the `MobileProcessClient` prompts for the IP address of the CNS {41 - 46}. The IP address and the global channel "A" {47} is used to create a connection to the CNS {48}. The following lines create the input and output channels to the server {49 - 51}. Once the `MobileProcessClient` receives a response from the server {52} a `ProcessManager` takes the `MobileProcess` and calls its `run` method {53}. This method call includes the concept of dynamic class loading as previously described.

What can be identified from the above code is the simplicity of the mechanism of agreeing a global connection. If all devices agree that the global connection to the CNS is "A" then each device can successfully establish communication with any other device on the network without requiring any additional steps.

2.5 Java

Sun Microsystems developed Java (2007d) after the phenomenon of Object Orientated programming languages such as SmallTalk and C++. The Java slogan is "write once, run anywhere" which is due to the compiler, the Java Virtual Machine (JVM). As long as the JVM is available on the system then the Java application should run without any problem. The JVM is currently installed in over 4.5 billion devices including personal computers and mobile devices.

The popularity of Java has established over 5 million Software Developers making it one of the most popular languages currently being used. To reflect this Java is now a mature language that has been refined, expanded and tested over the past 12 years resulting in the Java Application Programming Interface (API). The API is a collection of libraries and

packages containing classes that are ready to be used in any java application. The different libraries and packages include specialist areas such as security, sound, graphics and XML.

Java is also a static language providing efficient mechanisms to reduce errors as everything must be instantiated which is an obvious advantage during the development of complex systems. Another benefit is the backward compatibility with previous versions on Java. Currently on Version 6, previously built applications will still successfully run on the new platform.

Concurrency is another API available in Java using the concept of threads, which consequently is how Java is built and run, however the process of multithreaded applications is complicated. Once applications get complex, the java concurrency API becomes difficult and therefore an unpopular area in the Java framework.

2.6 Groovy

Inspired by the problems of Java and its static restrictions, Groovy (Codehaus Foundation, 2006a) has been created to give a seamless transition to Java developers to a dynamic typed scripting framework. The Groovy compiler works with the JVM to enable any Java code to be successfully compiled within a Groovy class. This offers the opportunity for developers to slowly make the transition to a dynamic language. Groovy uses similar syntax to Java and still represents the same object model.

```
55  One2OneChannel connect = Channel.createOne2One()
56  def processList = [ new Producer ( cout: connect.out() ),
57                      new Consumer ( cin: connect.in() )
58                    ]
59  new PAR (processList).run()
60  ____________________________________________________________________________
61  class Producer implements CSProcess {
62    def ChannelOutput cout
63    void run(){…}
64  }
```

Listing 6: ProducerConsumerMain Process and Part of the Producer Process written in Groovy

Listing 6 is Groovy code showing a main process instantiating `Producer` and `Consumer` processes {56, 57} and, after the horizontal rule {60}, code for the instantiated `Producer` process {61 - 64}. Listing 7 is the equivalent code written in Java,

The obvious difference is the amount of code written for Java compared to Groovy which achieves the same steps in nine lines as opposed to Java's seventeen. The type `def` {56, 62} enables variables to be dynamically changed in groovy so that they can represent

string and integer objects easily where Java would need to implement the code that does the conversion.

```
65  public class ProducerConsumerMain extends CSProcess
66    public void run () {
67      final One2OneChannel connect = Channel.createOne2One();
68      final CSProcess[] network = {
69          new Producer(connect.out()),
70          new Consumer(connect.in())
71      };
72      new Parallel (network).run();
73    }
74  }
75  ------------------------------------------------------------------------
76  public class Producer implements CSProcess {
77    final ChannelOutput cout;
78    public Producer(ChannelOutput cout) {
79      this.cout = cout;
80    }
81    public void run() {…}
82  }
```

Listing 7: ProducerConsumerMain Process and Part of the Producer Process written in Java

The ability to apply channel names within the `ProducerConsumerMain` process {56, 57} eliminates the need to define a constructor with the `Producer` class {62}. The following examples in Listing 8 show the characteristics of lists in Groovy.

```
83  [1, 2, [3, 4], 5]
84  ['Ken', 1, 2]
```

Listing 8: Lists in Groovy (Barclay & Savage, 2007)

The benefit of the above lists is that they are dynamic and naturally expand in size when new items are added. They also offer the opportunity to nest different types of collection like another list {83} or a map. They are also heterogeneous {84} and accept different types of variables to be stored.

Closures is another useful function introduced by groovy that is compared with the Java inner class. Using the following example, taken from Codehaus (2006b), the first part {85 - 97} is an example of how a class and method would be defined in Java and then called from another. The latter piece of code is written in Groovy {98 - 100} and has the same functionality as the Java code but uses a closure instead {99}. Line 100 declares that the value if `x` should be the first parameter in the closure, i.e. `numberTosquare` and the `c.call` requests that the closure `c` is executed and, as shown in the Java code, will return the result `4`.

```
85   package example.math;
86   public class MyMath {
87     public static int square(int numberToSquare){
88       return numberToSquare * numberToSquare;
89     }
90   }
91   ------------------------------------------------------------------------
92   import example.math.MyMath;
93   ...
94   int x, y;
95   x = 2;
96   y = MyMath.square(x); // y will equal 4.
97   ------------------------------------------------------------------------
98   def x = 2
99   def c = { numberToSquare -> numberToSquare * numberToSquare }
100  def z = c.call(x)  // longhand form, z will equal 4
```

Listing 9: Groovy Closures (Codehaus Foundation (b), 2006)

It can be seen that closures are a useful mechanism in groovy to reduce the amount of classes and lines of code. The additional benefit of closures is its use with collections as the code `list.each{some code to be executed here}` will apply the code within the brackets to all of the elements within that collection (`list`). This is used in Listing 6 {59} and is used for combining Groovy and SQL so iterating through results and the returned rows from a database is easier.

2.7 Grails

The creation of Ruby on Rails (Hansson, 2007) introduced the concept of dynamic language-based web frameworks. In 2005, inspired by this concept, Rocher (2006a) developed an open source version for the Java community called Grails. However one of Rocher's goals was "write less code and be Java friendly" so, in conjunction with Guillaume Laforge, the framework focused on the Groovy language. The principle "Convention over Configuration" (Rocher, 2006a) was used to ensure Grails provided an agile, effective framework for developing web applications. Using the "Don't Repeat Yourself" (DRY) principle Grails enables development at the first phase without thorough training or understanding of the overall framework's structure. Laforge (Rocher, 2006a) also describes learning Groovy as "an almost flat learning curve" for Java developers.

These principles have produced the framework shown in Figure 11 which remains object orientated but with a high level of abstraction. With the "Java friendly" goal in mind, the infrastructure of Grails was carefully selected to include the following open source technologies:

- **Hibernate** – Maps object-orientated objects to tables in a relational database. (Red Hat Middleware, 2006)

- **Spring** – Inversion of Control (IoC) framework which connects objects so that their dependencies are available at runtime.
- **Quartz** - Job scheduling framework including the completion of regularly or spontaneous tasks. (Open Symphony, 2000a)
- **SiteMesh** – Layout-rendering framework that helps create a consistent look for large complex sites hosted in grails. (Open Symphony, 2000b)

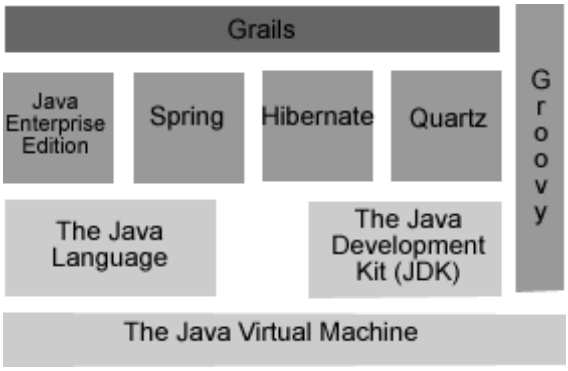


Figure 11: Grails Framework (Adapted from Rocher, 2006)

Grails uses the design architecture Model-View-Controller (MVC) as used for the creation of SmallTalk. This architecture supports the achievement of the principles described above.

> *"Grails is about providing an entry point for the trivial tasks, while still allowing the power and flexibility to harness the full Java platform"*
>
> (Rocher, 2006)

Grails is therefore providing a new and easier way for Java developers to create enhanced object-orientated web applications with the ability of making more complex systems, if required, using the integrated technologies highlighted above.

2.8 Bluetooth

Current mobile devices are enabled with wireless technology to remove the restrictions of cabled communication. Bluetooth, a replacement for infra-red technology, is intended for ranges of 10 meter radius or less. In conjunction with WiFi, mobile devices are capable of seamlessly connecting with other portable devices and computers without using cables or the necessity of being in line of sight. Bluetooth is not a primary technology for large scale connections as, compared to WiFi, is slower and has weaker signal strength.

Therefore it is ideal for peer to peer communications as it will not interfere with traditional communicational methods used for television, telephones and radios.

Bluetooth was developed by a user base consisting of leading companies such as Ericsson, Intel, Nokia, Toshiba and IBM (Bluetooth SIG, 2007). Averaging 1 – 3mbps, Bluetooth is useful for small scale applications enabling eight devices to connect simultaneously without any interference due to spread-spectrum frequency hopping (Bluetooth, 2007). Bluetooth uses up to 79 different frequencies separated by different device types therefore the likelihood of the same frequency being used is very limited. This feature enhances the security of Bluetooth which is commonly scrutinised as it is not as secure as other wireless technologies such as WiFi. Spread-spectrum frequency hopping enables Bluetooth to regularly change frequency channels without disruption preventing regular surveillance of the same channel.

When two devices connect using Bluetooth they create a piconet or personal-area network (PAN). Using frequency hopping, the two devices regularly change frequencies but remain connected by intelligently avoiding other piconets that may exist in the same 10 meter range.

2.9 Apache Derby

Apache Derby is an element of the DB Project by the Apache Software Foundation (Apache Software Foundation, 2007) who specialise in maintaining and developing open source database solutions. Derby is developed in Java and available as a useful relational database. It can be easily integrated with Java and Groovy applications and is therefore portable and able to maintain performance. In 2001 IBM donated the source code of their Cloudscape database system to start up the Apache Derby project. In 2005 an official release of Apache Derby DB was launched with positive appraisal that Sun Microsystems are now using a secure version for their Java DB project (2007c).

Apache Derby DB implements the core Java Database Connectivity (JDBC) classes within the java.sql package to enable programs to connect to the database using the JDBC drivers. It is available as a client-server or embedded version. The embedded version runs on the same JVM as the program and is stored on the local file system. The client-server version works on a different JVM to the program and uses a connection over a network to communicate.

2.10 FreeTTS

Inspired by the development of Flite by Carnegie Mellon University, FreeTTS (Text to Speech) is a speech synthesis system written in Java. Built by the Speech Integration Group at Sun Microsystems (2005), FreeTTS is available as an open source library. The framework is capable of creating audio files by converting ASCII text into spoken vocabulary and is then capable of playing the audio files. FreeTTS lacks documentation and only contains a selection of demonstrations to give an idea of its capabilities.

3 Design Requirements

This chapter discusses the expectations of the application in terms of the visitor and museum. Although discussions have concentrated on the visitor's role and needs for the application, the museum itself will have expectations to easily maintain and manage the technology once it is installed.

3.1 PACT Analysis

Human Computer Interaction (HCI) first came to prominence in the 1980s to assist the development of technology that was to be human centred. Focusing on the design, evaluation and implementation aspects of development, HCI provides a step by step assistance to ensure the system suitably interacts with people within their everyday lives. Benyon, Turner and Turner (2005) discuss a framework with 4 key principles for designing a successful HCI application; *People, Activities, Context and Technology (PACT).*

Analysing these principles individually should identify the key elements and areas of concern that will successfully assist people in using the interactive system. Mentioned as materials in Benyon et al (2005), the elements for this project are; Visitors, Technology and Museum. Each element needs to be considered as part of the overall design for the project and each element has different requirements to the other but has an affect on how the requirements will be implemented. Understanding the available technology is crucial but without knowledge of the museum environment the technology could be too large or expensive for the project's budget. Developing an application using the technology that meets none of the expectations of the visitor is an unsuccessful project so the visitor requirements should also be highlighted. Figure 12 is a diagrammatic representation of the above relationships.

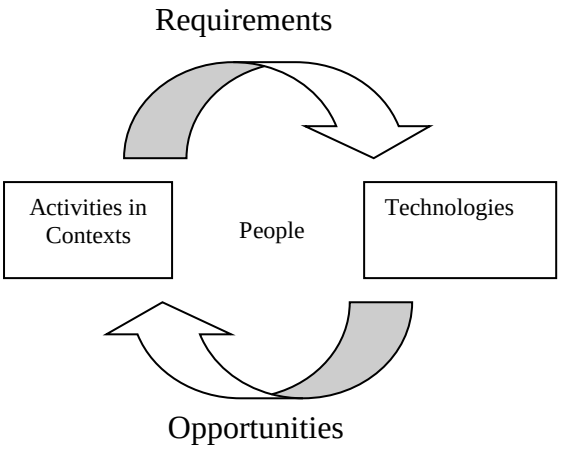


Figure 12: The Relationships in PACT

The following statement shown in Figure 13 is the main objective to be achieved by this project. The statement has been separated into its corresponding principle within the PACT framework. Each principle will be individually discussed and considered in the following sections and should highlight a combination of design requirements that should be used as part of the technology design and later, the implementation, testing and user evaluation.

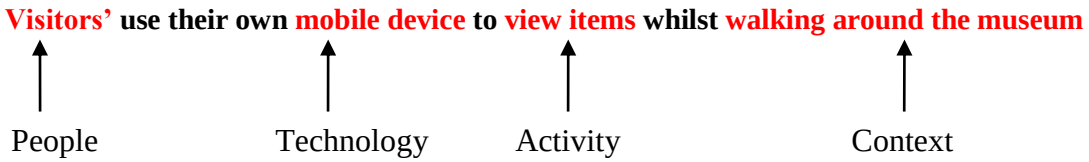


Figure 13: Using PACT principles to identify the concepts

3.1.1 Human Centred Design

There are two approaches to developing interactive applications for humans; Human or Machine centred. Machine centred developers fail to consider the ease of using the finished product. Humans will therefore not use it or continuously refer to a user manual. HCI and subsequent additions are trying to prevent this failure to ensure the product is fit for purpose and use.

Safety ensures the interactive device is easy to use and understandable. Safety critical systems have failed due to the design of the user interface. Buttons are confusing or hidden by other parts of the system resulting in failure and sometimes fatalities. Effectiveness measures how the system interacts with everyday lives and observes any

disruptions or reduced periods of productivity. Ethics concentrates on how the user feels about the system and whether they feel it is trustworthy. Consider an ATM machine where the safety could relate to the user and their surroundings, the effectiveness being the time duration for a transaction and ethics is how the user feels about their account details being used to obtain balance information. These measurements will be revisited during the user evaluation in Chapter seven.

3.1.2 People

Everybody is different and has their own strength and weaknesses' therefore designing a system to suit everybody is extremely difficult and expensive. Taking the senses as an example, sight alone results in numerous considerations including colour blindness and short and long sightedness. Hearing impairments is another area for consideration if we intend to use sound within the system. Memory loss or learning difficulties should also be thought of if relying on long instructions or guidance notes. Different cultures use symbols or colours to mean different things so care needs to be taken when using these as part of the user interface.

Therefore this principle has ascertained that it would be beneficial to display the information in a number of different formats to ensure all users have the ability to use the system. Displaying images that are exact representations of the item, providing textual descriptions but also play the same as a sound for the visually impaired.

Advantageously the visitor will provide their own mobile device which, if compatible, will download the required software. Therefore general device interface issues are non problematic as the visitor should be comfortable and familiar with their own device. However to ensure compatibility, the user interface must maintain a basic skin and functionality that the user can apply successfully to their own device.

An advantage of using the mobile process technology for the system is that once the visitor leaves the museum, or is out of range of the application, the software will be removed from their mobile device.

3.1.3 Activities

The project's main activity is to allow visitors to view items within the museum using their own mobile device. However there are many smaller issues that can be ascertained

by this bold objective. The Activities has been divided into smaller characteristics to enable developers to think thoroughly about how the activity will be undertaken.

- ***Temporal*** – how often is the activity to be under taken, will multiple users interrupt the system, what happens if the system is interrupted, is there a logical process to the activity and what should be an expected response time.
- ***Cooperation*** – Is the activity an individual or group task and how do the two differ.
- ***Complexity*** – Is there a sequential process to the activity? Are there several tasks that are not interrelated? Can the user return to a previous section easily?
- ***Safety Critical*** – Can the system recuperate from a user's mistake and are there any other risks involved (including an injury or accident).
- ***Nature of Content*** – data requirements, hardware requirements and expected input and output mechanisms.

In terms of temporal aspects, the system should provide a simple mechanism for viewing museum items in a variety of different media. The user may select an item from a selection, as one would expect in a museum cabinet, and then view the sound/text/image of the chosen item. The user should then be able to return to the previous item selection page as the complexity aspect highlights. Cooperation is an interesting point to consider as it is unlikely for all members of a family to own a mobile device therefore it would be useful to allow a number of users to view the item information at one time. Safety critical may not be an immediate aspect to ponder for this project but still needs to be considered. An accident that may happen to a visitor when using the system is the lack of peripheral vision. Therefore obvious consequences are bumping into other items, missing a step or annoying other visitors. The system must intrigue the user but only distract them for small periods of time.

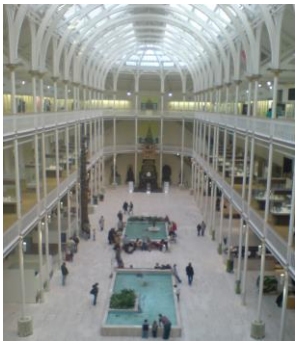
3.1.4 Context

Context takes the activities described previously and applies them to the physical environment of the museum and is divided into three sub sections:

- ***Physical*** – focuses on the actual environment that the user will undergo the activity.
- ***Social*** – will the user work alone or as a group and should assistance be available.
- ***Organisation*** – what impact this will have on the museum including job allocations and customer satisfaction.

The National Museum of Scotland in Edinburgh is a popular tourist attraction combining new and old architecture together to display a vast array of exhibitions. The museum will be used as the example for this project. The older building is divided into 3 floors with a large hall in the centre with access to several smaller rooms where the exhibits are displayed. Each room displays their artefacts differently with no common layout in use. All rooms have high ceilings and some lack lighting to improve the ambience.

Modern Building



Old Building

Figure 14: Photos of the Museum and the different building structures

The modern building is divided into 6 floors with a mix of high and low ceilings. Light is a key element of the building's design so contrasts with the older building. Some artefacts are large and appear in more than one floor due to the height.

As mentioned in section 3.1.3, the project should not focus on group or individual visitors but provide a mechanism so both are capable of enjoying the system. Assistance may be available but only for the user interface and not the use of the mobile device as the visitor should already be familiar with its functionality.

The system will have some impact on the museum during the initial setup of the system whilst the technology is installed and the museum items stored in different media formats. Thereafter staff will need to be familiar with the system so they can update details of the items but should be an infrequent task.

3.1.5 Technology

The previous sections have highlighted some of the main visitor and museum requirements and considered the system environment. The Technology principle of PACT identifies the expected input and outputs (I/O) from a human perspective. The visitor is expected to use their own mobile device to interact with the UI however the

requirements and expectations need to be elaborated to make the foundations used within the software design process.

The project will use the Unified Modelling Language (UML) principles to identify and design the software for the project. The first stage is to list all the expected visitor and museum requirements which are known as use cases. The use cases will then be used during the development process of the software and regularly to ensure the project is keeping within its requirements.

3.1.5.1 Use Cases

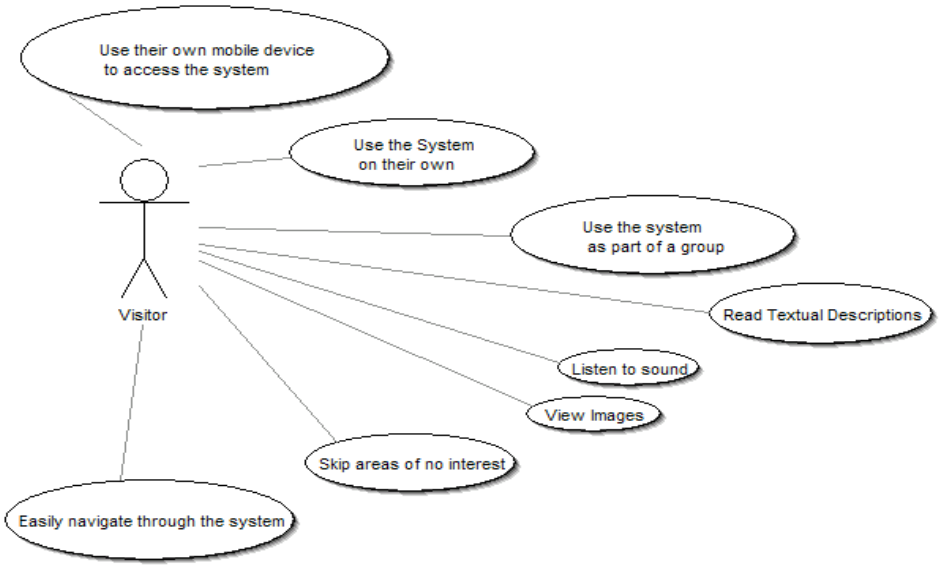


Figure 15: Use Cases for Visitors

UML uses the concept of *use cases* to list all the possible user requirements in a diagrammatical form. Figure 15 is an example of a *use case* diagram displaying eight different *use cases* for the museum visitors. When developing and testing the prototype, the above *use cases* will be revisited to determine if the requirements have been met. Another useful aspect of this concept is that each *use case* can be simply transformed in to a *test case* and will help to ensure that all functionality is tested.

Visitors are not the only users as maintenance of the system is crucial and must be considered. Museum staff will need to have the ability to modify the system and adapting the data to reflect exhibit changes. Figure 16 is the *use case* diagram for the actor, museum staff. There are three key areas that will be managed by this user; museum items, access points and exhibition areas.

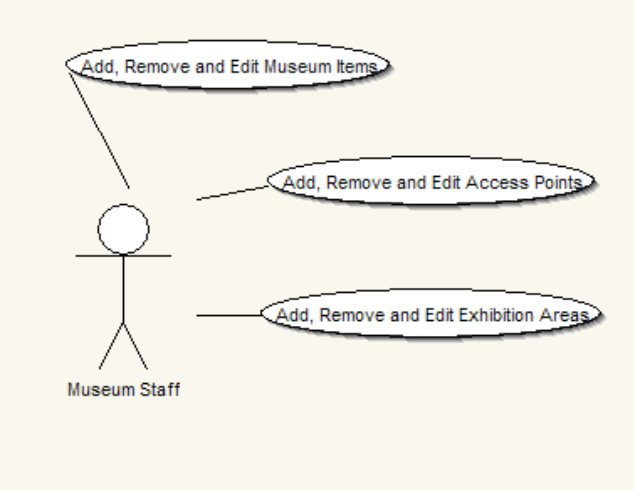


Figure 16: Use Cases for Museum Staff

Museum items are usually divided into exhibitions areas, for example Egyptians, Mammals or Sport. To implement the system, items will need to be associated with an access point that will then return a selection of the various items available to the visitor.

3.1.5.2 Sequence Diagrams

The *use cases* can now be used to determine the interactions between the different components of the system and help identify the various features that are required. Sequence diagrams can show steps chronologically and highlight where different components of the system communicate or carry out an activity.

Figure 17 is an example of a sequence diagram showing the steps involved in creating a connection between the museum application and the visitor's mobile device. Using the JCSPNet features mentioned in Section 2.4.2 when the user enters the museum their mobile device will automatically detect the Channel Name Server (CNS). The CNS will then establish the connection between the Mobile Device and Museum Application. The Museum Application will then download a process using JCSPMobile so a graphical User Interface (GUI) will appear on the visitor's device with the option to start or exit the application. In the sequence diagram, the visitor decides to start the application resulting in the Museum application querying the database for details of items that are currently in range. Details of these items will then be returned to the Museum application and, in a suitable format, sent to the mobile device to be displayed to the visitor.

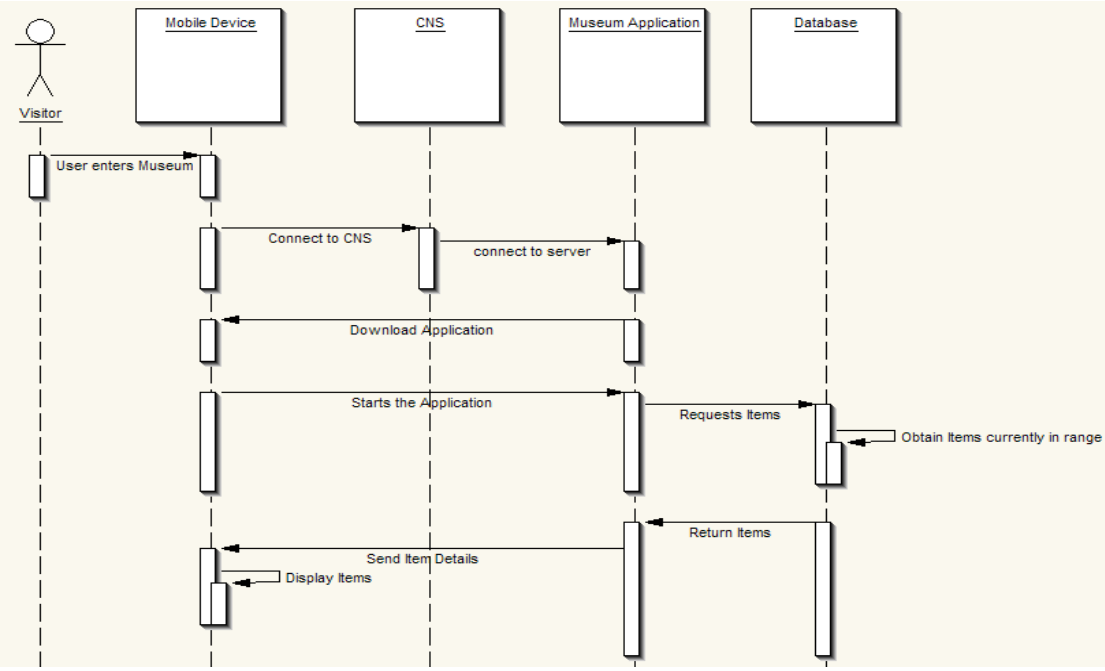


Figure 17: Sequence diagram for initial interaction of user and system

Figure 17 identifies the use of a central repository for managing the information of museum items. As identified in Section 3.1.5.1, museum staff will be responsible for updating the same database when modifying the information of museum items, access points and exhibition areas.

Grails is built as an OO web application so Figure 18 is an example of the sequence diagram for museum staff adding a new exhibition area called “Mammals” with a new access point “1” and a museum item called “Whale”. The structure of the objects item, accesspoint and exhibition will be reflected in the database which is a useful characteristic of Grails known as Grails Object Relational Mapping (GORM) which will be discussed in detail in Chapter 5.

Adding new information as shown in Figure 18 can take up to three steps with the database updated when necessary. First the user will create a new Exhibition and second create a new Access Point which will be associated with the Exhibition. This is a useful OO relationship where the Access Point will only be available in this Exhibition area so reduces duplication and eases management of the overall structure of the application. Within a database this relationship is known as a foreign key. The same process applies when adding the museum item which is associated with an access point.

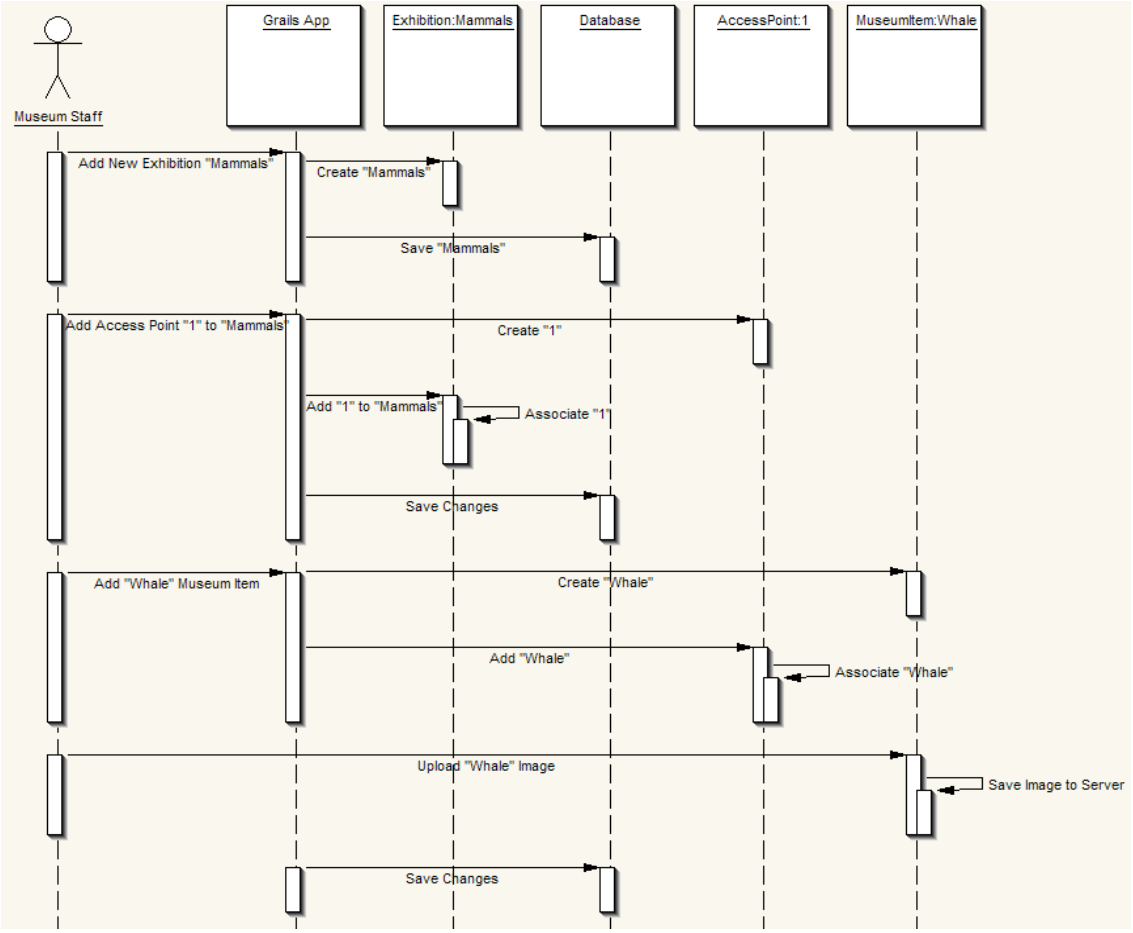


Figure 18: Sequence Diagram for Museum Staff

3.1.5.3 Object Diagram

OO design uses abstraction to represent the real world components. Concurrency however is not object based, but instead, process orientated. Therefore the third step of the UML process is to identify the various objects and relationships within the system. The following figure is a representation of this step using processes as the objects and the connections between these processes are the communication channels.

In the above steps we have identified the following components for the visitor application; Mobile Device, CNS, Museum Application and Database. Using the mechanism of JCSPNet we can elaborate that initial diagram shown in Figure 9 to the one shown in Figure 19.

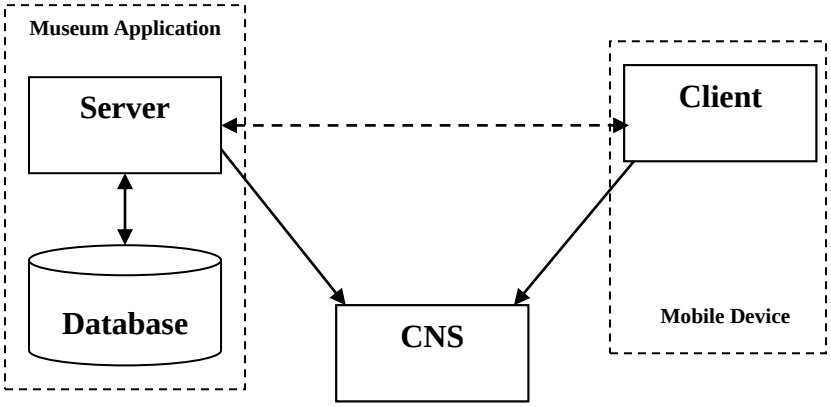


Figure 19: Process Diagram of Museum Application

As mentioned Grails still remains object orientated so Figure 20 is the class diagram for the application for the museum staff. The diagram shows the various attributes each object possesses and the associated relationships between each type of object.

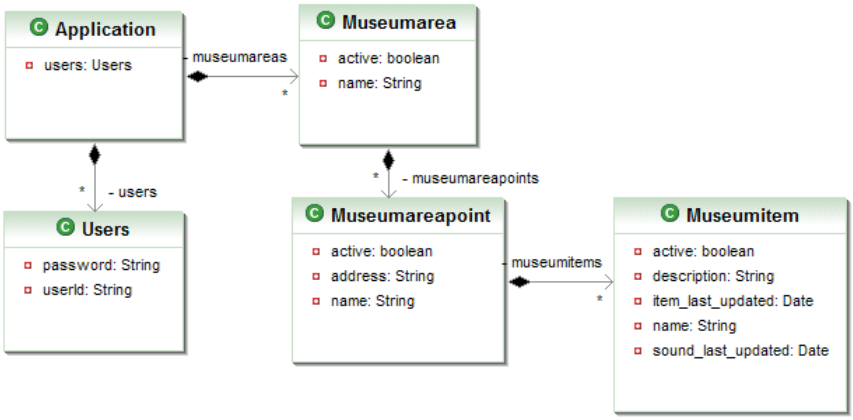


Figure 20: Class Diagram for Grails Application

The Application class represents the main Grails interface and is added to show the 'many' relationship for the Users and Museumarea classes. The Users class will be the registered members of staff that can login and access the system with a unique user ID and password.

The Museumarea has a name identifier and a Boolean value. Museumitem and Museumareapoint also have Booleans called 'active' to enhance usability. If an item or exhibition area is closed, the system can switch the Boolean rather than deleting all the files. Museum items can go on special loan for limited periods of time so it is useful to have this Boolean mechanism to manage such scenarios.

The associations between the classes show the relationships between them so Museumarea will have a collection of museumareapoints. So in the real world a museum area such as Mammals may be divided into different sections such as Whales or Seals. Having separate area points, i.e. wireless access points, restricts the user receiving too much information at one time and group similar items together.

3.1.6 Data Storage

As mentioned in Section 3.1.5.2 Grails uses the concept of GORM. The classes shown in Figure 20 are known as Domain Classes in Grails. So taking the User domain class as an example once created GORM, and the underlining technology Hibernate (Red Hat Middleware, 2006), will create a table in the database called Users with four columns; version, id, userId, password. Version and id are required by Hibernate so Grails automatically adds these fields to any domain class. The id field is used as the private key and is unique for every record. Apache Derby's database can be easily integrated with Grails (Tang, 2007) by changing the data source files within the conf directory of the Grails project. Therefore any created domain classes or changes to existing data will be automatically updated in the database without any additional coding required.

The visitor application can use Groovy to query and extract the required data from the database without affecting the Grails application.

4 Methodology & Implementation

This chapter focuses on the development process of the interactive application. Initially identifying development methodologies and in particular highlighting the suitable method for the project. The chapter then uses the chosen methodology to apply it to the implementation of the application.

4.1 Prototyping

Incremental development as shown in Figure 21 is a useful implementation method as the requirements of the system are divided into stages. Each stage is then designed, implemented and tested. If a previous stage is available then the new one is integrated into the system and all functionality is tested.

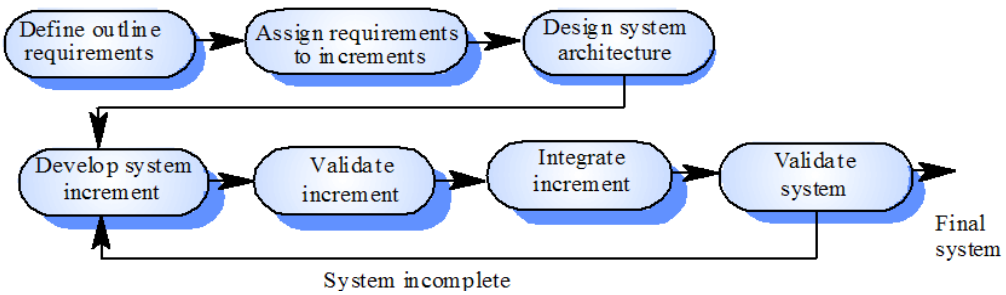


Figure 21: Incremental Development Flow Diagram

Each increment represents new functionality for the system and is prioritised by importance so crucial features are developed first. This workflow enables the critical parts of the system to be tested numerous times during the development cycle. At the end of each stage, or increment, a prototype should be available. This technique reduces the risk of project failure as each increment determines or changes the requirements for the following increment. If the next increment causes issues then the project should be able

to roll back. The following sections will discuss the implementation of each of the increments and with it the new functionality.

4.2 Initial Experiments

4.2.1 JCSP Mobile

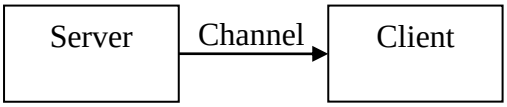


Figure 22: Server-Client processes with one channel

Development began by trying to understand the concept of a Process Orientated environment including the processes and message passing approach using channels within the JCSP framework. A key element to Software Engineering is the Gamma, Helm, Johnson, & Vlissides (1995) Design Patterns therefore a suitable starting point was the concept of the Client-Server, a key element to process orientated design. The program consisted of the `Server` sending text to the `Client` via a `Channel` which was then output to the Integrated Development Environment's (IDE) `console`. Figure 22 is a diagrammatical representation of a process orientated system where two processes; Server and Client communicate via a channel represented as an arrow. The direction of the arrow shows the flow of message passing so in Figure 22 the Server communicates with the Client. This test worked on a single Java Virtual Machine (JVM) so the natural progression was to extend this implementation to include JCSPNet as shown in Figure 23 so the system could communicate over a TCPIP network.

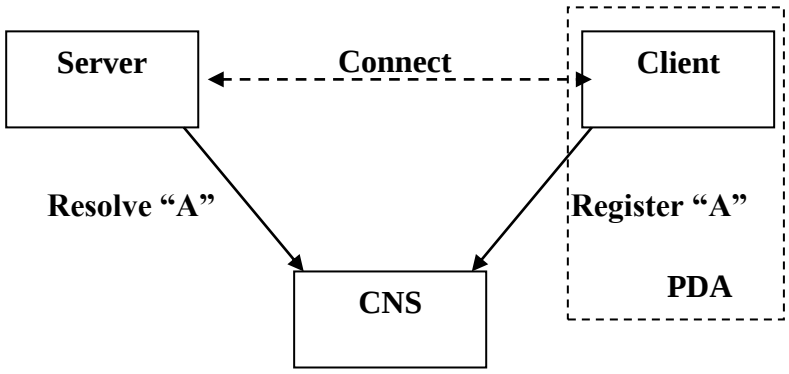


Figure 23: JCSPNet implementation for the Client-Server processes

The Channel Name Server (CNS) is responsible for managing the communication between the different processes on a network. Therefore each process is aware of a global channel called "A". The Client and Server processes register their addresses with

the CNS by sending details of their location onto channel "A". The CNS can then send details to the client of a server that it can connect to and hence a temporary channel called "Connect" is created to enable the Client-Server to communicate.

Listing 10 is an example of the client process which uses the global channel "A" {101} to communicate with the CNS {102}. The client then creates and output channel toServer to the CNS server using the location returned by the global name A {103}. The client then creates a channel processReceive {104}, which will be used as the input channel by another process, and sends the location of this channel to the CNS {105}.

```
101  String processService = "A";
102  NetChannelLocation serverLoc = CNS.resolve(processService);
103  NetChannelOutput toServer =NetChannelEnd.createOne2Net(serverLoc);
104  processReceive =  Mobile.createNet2One();
105  toServer.write(processReceive.getChannelLocation());
```

Listing 10: Connecting to global channel "A"

4.2.2 JCSP Mobile

As mentioned in Section 2.4.3 JCSPMobile has enabled the ability for processes to be downloaded to another device, using dynamic class loading, via a TCP/IP connection and then successfully run. Once a client has communicated with the CNS and created a connection with the Server, the Server will try to send a new process to the client.

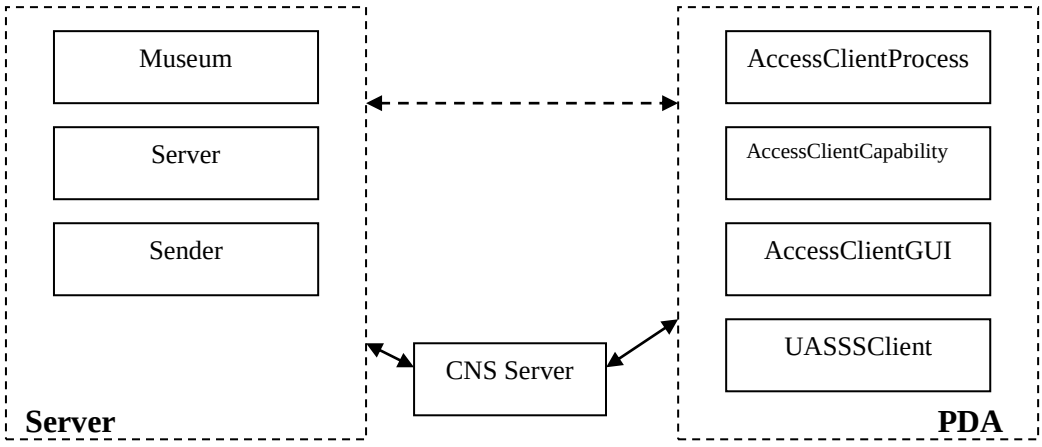


Figure 24: Museum application implementing JCSPMobile

Figure 24 is an example of the museum application and the initial processes downloaded to the PDA once the main channel communication has been established. AccessClientProcess is shown in Listing 11 detailing the partial creation of some of the channels {109, 110} and the initiation of some of the processes {112, 113}. The network array {111} contains two processes called AccessClientCapability and

`AccessClientUserInterface` {112, 113}. These are not currently available on the mobile device so when the entries in the array are run {115}, JCSPMobile will download the two processes to the device using dynamic class loading. This enables efficient running and optimal performance of the application, where space is limited, because only what is required is downloaded to the device.

```
106  public class AccessClientProcess extends MobileProcess
107  {
108    public void run () {
109      final One2OneChannel A2G = Channel.createOne2One();
110      final One2OneChannel G2A = Channel.createOne2One();
111      final CSProcess[] network = {
112          new AccessClientCapability (…),
113          new AccessClientUserInterface (…)
114      };
115      new Parallel (network).run();
116    }
117  }
```

Listing 11: AccessClientProcess running two processes

The following sections will discuss the incremental development of the application in more detail and further diagrams will show the mobile device with a main Client process, for clarification this represents the UASSSClient, AccessClientProcess and AccessClientCapability processes.

4.2.3 Graphical User Interface

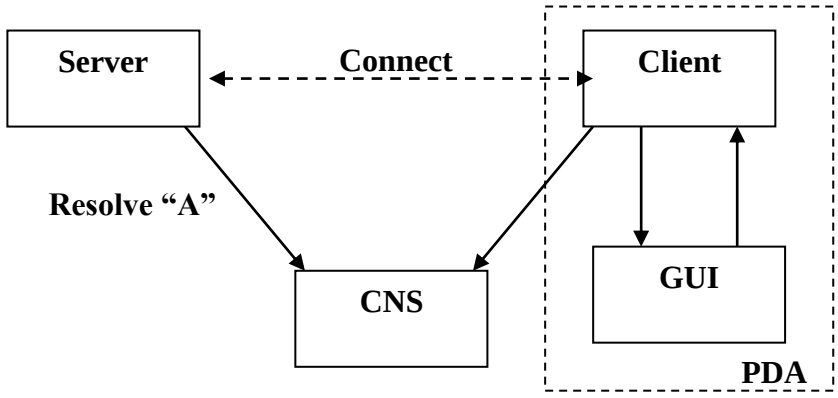


Figure 25: Adding a Graphical User Interface Process

The JCSP framework has extended the java.awt API to enable awt classes to be implemented using channels. This additional functionality enables the ability to add channel implementations so processes can read and write data from the different graphic components. Figure 25 is an example of the process diagram with the additional process of a `GUI` communicating with the `Client` process. The `GUI` process is the same as the `AccessClientUserInterface` process discussed in section 4.2.2. Figure 26 is an

example of an `ActiveButton` from the `JCSP.awt` framework (Welch & Brown, 2007). The button is an extension of the `java.awt.Button` class and as shown implements several different channels. Messages are sent down the configure channel to set up the component and additional event channels can be implemented before the process is run. The Event channel sends the current value of the button's label after it is pressed.

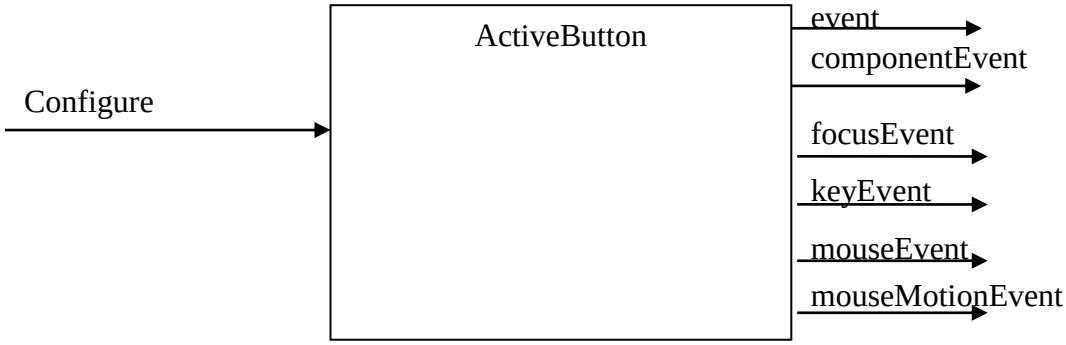


Figure 26: Example of a jcsp.awt.ActiveButton class and channels

The GUI requires different functionality to display text, manage sound and show one or more images. The process diagram in Figure 25 has been simplified to exclude the various channels required to implement the different components provided by the `jcsp.awt` framework.

Listing 12 is an extract from the `GUI` process showing the creation of an `ActiveTextArea` {118}. Similar to the `java.awt.TextArea` the `jcsp.awt` version accepts an input channel `textFromConsole` which is the text sent from the `Client` process. The latter three parameters are for the width, height and scrollbars respectively. This component is used in the application to display instructions and details of the museum items.

The final part of the Listing {127 - 135} is the creation of the process for the `GUI` so that all active components are added to a CSProcess `network` and started together {135}. This ensures that all components are running and listening for any interaction from the user via the `GUI`. As discussed, when a component encounters interaction from the user it will inform the `client` process. However this is only relevant if a suitable output channel has been passed as a parameter.

```
118  final ActiveTextArea text = new
119  ActiveTextArea(textFromConsole,null,null,3,1,1);
120  final ActiveButton newButton = new ActiveButton ( pauseButtonConfig,
121  buttonEvent, "START" );
122  final ActiveButton closeButton = new ActiveButton (closeButtonConfig,
123  buttonEvent, "EXIT");
124  final ActiveTextEnterField inText = new ActiveTextEnterField
125  ((AltingChannelInput) clearInputArea, toClient);
126  actionContainer.add(inText.getActiveTextField());
127  final CSProcess []network = {
128                       root,
129                       text,
130                       images,
131                       inText,
132                       newButton,
133                       closeButton
134         };
135  new Parallel ( network ).run();
```

Listing 12: Example of a GUI process

Another useful aspect of `jcsp.awt` is the ability to monitor button events. Lines {120 - 123} are the creation of two `ActiveButton` components called `newButton` and `closeButton`. `pauseButtonConfig` and `closeButtonConfig` are two input channels connected to the client process which provide the ability to dynamically change the text of the button when required. Output channel `buttonEvent` {123} informs the `Client` process of any button presses from the `GUI` and is the main function for the user to navigate around the application.

The `configure` channel is responsible for setting up the component and has a corresponding inner class that is invoked every time there is an interaction. As a default, the `ActiveTextArea` {118} will append any new text sent via the `textFromConsole` {119} channel. As the museum will be displaying instructions and information of different items it would be beneficial to clear the text area before a new entry is added. Therefore Listing 13 is an example of a redefined `ActiveTextArea` called `MuseumActiveTextArea` {136}. It implements the `Configure` inner class and redefines the `configure` method {141} to additionally call `setText` method {142}.

```
136  public class MuseumActiveTextArea implements ActiveTextArea.Configure {
137   public String message;
138   public void setMessage(String s){
139         this.message = s;
140   }
141   public void configure(TextArea ta) {
142         ta.setText(message);
143   }
144  }
```

Listing 13: MuseumActiveTextArea class

The `ActiveTextEnterField` {124} is a redefined `java.awt.TextField` component to enable a listener to notify a process, in this example the `Client`, of the return key being

pressed whilst in this field. Listing 14 is an example of the `Client` class listening for an action from the `ActiveTextEnterField`.

```
145  int selection = Integer.parseInt((String)fromConsole.read());
```

Listing 14: Listening to a ActiveTextEnterField

4.2.4 Image

As the user is moving around the museum there may be more than one item available to view at any given time so there needs to be an efficient method of displaying these items so the user can easily identify and select the required artefact. Figure 27 is an updated diagram showing the introduction of two new processes `Image` and `ImageServer` and a class `ByteBuffer`.

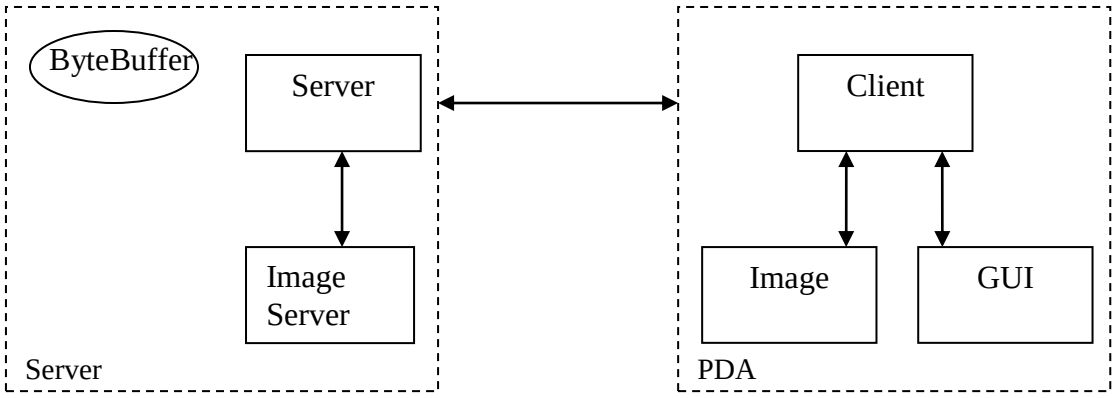


Figure 27: Updated system diagram with Image capabilities

In the initial design it was thought beneficial to have a mechanism that all media required for the application should be translated to a byte array and then passed using a serializable object. This would ensure that the data passed would not be corrupted and the client side would be able to use a simple mechanism to distinguish the different types of data.

```
146  public class ByteBuffer implements Serializable {
147   private byte[] b;
148   private int type = 0;
149   private int id = 0;
150   public ByteBuffer(int type, byte[] b){
151         this.type = type;
152         this.b = b;
153   }
154  }
```

Listing 15: ByteBuffer class

Listing 15 is an example of a `ByteBuffer` class that stores images, sound and text. It implements the class `Serializable` {146} so the object can be transferred over a network connection. The `ByteBuffer` has three main attributes; `b` {147}, `type` {148}

and `id {149}`. The `byte[], b` stores the media and type is used by the `Client` process to determine the type of media, i.e. sound or image. The `id` attribute stores the unique id for the item as logged by the museum and is used by the `Client` to request details of a particular item when required.

The `ImageServer` process is responsible for reading the images from the file server and converting the details into a `byte[]` as shown in Listing 16. The images are read in as a stream {161 - 162} and then converted to a `byte[]` {164 - 170}. The values are then assigned to a new `ByteBuffer` {158, 159, 171, 172} and passed to the main `Server` class {175}.

```
155  Class ImageServer implements CSProcess {
156    void run(){
157      def image = imageRequest.read()
158      ByteBuffer bb = new ByteBuffer()
159      bb.setID(Integer.parseInt(image))
160       try{
161          InputStream is =
162  this.getClass().getResourceAsStream("images/${image}"+".jpg")
163          if (is.available() != null) {
164             BufferedInputStream bis = new BufferedInputStream(is);
165             ByteArrayOutputStream baos = new ByteArrayOutputStream();
166             try {
167                int ch
168                while ((ch = bis.read()) != -1) {
169                  baos.write(ch)
170                }
171                bb.setB(baos.toByteArray())
172                bb.setType(1)
173             } catch (IOException exception) {…}
174        }catch(Exception e){…}
175       sendImage.write(bb);
176    }
177  }
```

Listing 16: ImageServer process

The `ByteBuffer` that is created is then wrapped within a main `Serializable` object called `MuseumData`. `MuseumData` stores details of all the `ByteBuffer` objects and the communication details such as channel address, client id and the request type. The latter attributes enable efficient communication between the `Client` and `Server` processes. The `requestType` attribute determines whether the items returned are a list of thumbnails, an unavailable area or details of a single item. The `clientID` attribute distinguishes the requested client with others on the network and the `channel address` determines where responses should be sent to once the data has been received.

Listing 17 is an example of an `ActiveCanvas` {178} which is a JCSP revised version of `java.awt.canvas` and therefore an active graphics component. The `ActiveCanvas` is used to display the thumbnail and individual images of the different museum items. It

expects an array list, dList {179}, as a parameter. The parameter dList is an instantiation of a DisplayList which contains a collection of GraphicsCommand objects.

```
178  final ActiveCanvas images = new ActiveCanvas();
179  images.setPaintable(dList);
180  textContainer.add(images);
```

Listing 17: ActiveCanvas in the GUI process

The GraphicsCommand is also an array that stores different graphics components that relate to each other. Listing 18 shows use of a GraphicsCommand {182, 185} and its benefits as it implements all the methods available in the java.awt.graphics package. The processImage method {192 - 210} uses the drawImage method {197, 202} inherited from the java package to add the Image to the GraphicsCommand array. The processImage takes the ByteBuffer from the main Client process and uses the byte[] contained within the object to create an Image object using {193} the Toolkit class available within java.awt. This class enables a byte[], originated from a JPEG or GIF image format, to be passed as a parameter and converted to an Image object without saving any data onto the client's device. The Image object is then added to the GraphicsCommand {197, 202} with the additional information of its x and y axis points which relates to where on the ActiveCanvas it should appear. The last part of the processImage method is the additional entry of some text that will appear under a thumbnail {206 - 207} so users will know what number to enter in their selection.

Once the GraphicsCommand has been populated the DisplayList has a method, set {189} that will take the array and request it to be executed through the ActiveCanvas. The ActiveCanvas will take the array and execute the static method paintable {179} so the images are written to the canvas during the GUI creation. ActiveCanvas does have the additional channels of toGraphics and fromGraphics which will dynamically update the canvas. To finalise, the Image process will return to the main Client process for further processing before the new GUI is loaded with either the thumbnail selection or item screen with one large image.

```
181   public class AccessClientImage implements CSProcess {
182     private GraphicsCommand [] museumGraphics;
183     private DisplayList dList;
184     public void run() {
185       museumGraphics = new GraphicsCommand [ 2];
186       museumGraphics[0] = new GraphicsCommand.ClearRect (0, 0, 300, 200);
187       processImage((ByteBuffer) images.get(0),1, false);
189       dList.set(museumGraphics);
190   … }
191
192     private void processImage(…) {
193       Image img = Toolkit.getDefaultToolkit().createImage(buffer.getB(),
194                   0, buffer.getB().length);
195       if(thumbnail){
196         …
197         museumGraphics[i] = new GraphicsCommand.DrawImage(img, (x),
198                             (y),50,50);
199       }else{
200         x = 0;
201         y = 0;
202         museumGraphics[i] = new GraphicsCommand.DrawImage(img, (x),
203                            (y),150,120);
204       }
205       if(thumbnail){
206         museumGraphics[images.size()+i]= new GraphicsCommand.DrawString(
207                                          "Number " + i,x,(y-10));
208         x = x + 60;
209       }
210     }
211   }
```

Listing 18: AccessClientImage process showing the benefits of the GraphicsCommand class

4.2.5 Database

The museum contains a variety of different areas with numerous artefacts within each so there needs to be an effective way to manage the information. In addition to this the new features of the system including the relevant images, sound and text will need to be stored and referenced therefore a useful tool would be a central repository such as a database.

This section will focus on the implementation of the database through the visitor's system. However the Grails application will be responsible for updating the data stored in the central repository and will therefore be discussed in the following Chapter.

Visitors may also want to download information about certain items and at times pick a specific item from a selection. To enable such interaction the application needs to interact with the Database that stores all the museum information. Figure 28 shows a diagrammatic representation of the updated system. It is shown that the PDA side of the application is not changed due to this new facility but a new process called the `DatabaseServer` has been added to the Server side.

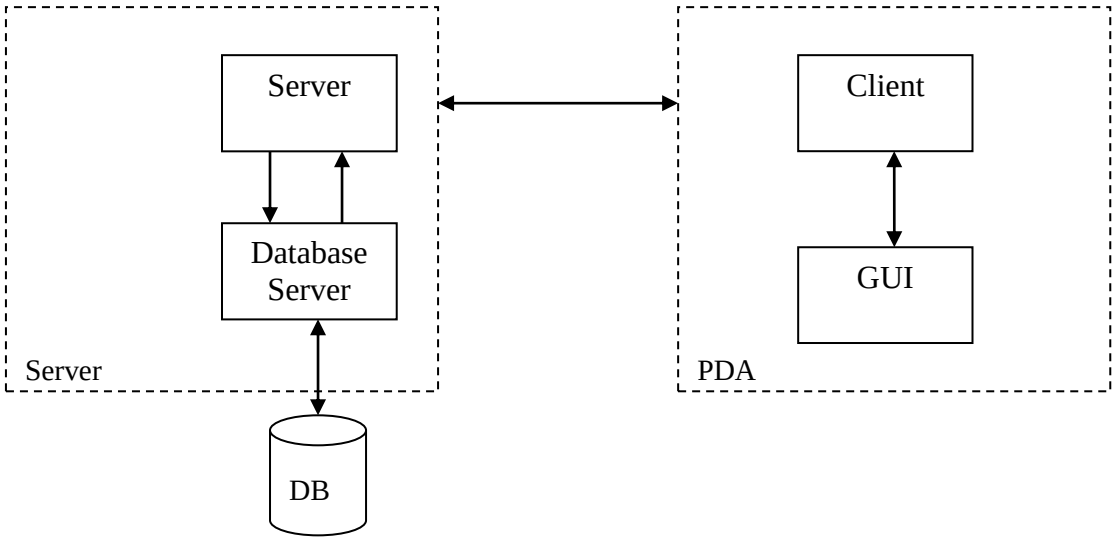


Figure 28: Updated Application Architecture with new Database Process

Listing 19 is an example of Groovy code that connects to an Apache Derby database {224} using parameters such as `DB` {213} where the database is stored, `USER` {214} and `PASSWORD` {215} are used to verify the connection and add security to the database. The latter parameter is the `DRIVER` {216} which determines the type of database, for example the `ClientDriver` runs Apache on a separate JVM and therefore enables the Grails application to share the same database.

Within the `run` method of the `DatabaseServer` a new connection to the database is instantiated {224} called `sql`. This connection can then send various request including delete, add, update and insert. In Listing 19 there is an example of a simple select request to the database where `museumitem` and `museumareapoint` tables {231 - 236} are queried. The groovy code `sql.eachRow` {231} is an example of a groovy closure as described in Section 2.6. This code ensures that every record within these tables is included in the search and for each valid record their name is output to the console {233 - 234}. The `?` {233, 232} represent placeholders for the parameters defined within the `[]` brackets. The `location` parameter relates to the CNS IP address that is stored in the `MuseumData` object that is passed by the server {226, 228}. The `active` field ensures that only ones with entry `1`, i.e. yes for being active, are retrieved. The last part of this SQL statement prints the name of the museum item and adds one to a counter called `answer` {235}. This SQL statement is executed at the beginning of a users connection to determine if there is more than one item available for viewing in the chosen area and will therefore decide if a

collection of thumbnails are returned {238} or an image, sound and text for a single item {237}.

```
212  class DatabaseServer implements CSProcess {
213    def DB = 'jdbc:derby:/museumDB'
214    def USER = 'museum'
215    def PASSWORD = '12345'
216    def DRIVER = 'org.apache.derby.jdbc.ClientDriver'
217    def ChannelOutput sendItemDetails
218    def ChannelInput serverRequest
219    def ChannelOutput getImage
220    def ChannelInput receiveImage
221    def ChannelOutput getSound
222    def ChannelInput receiveSound
223    void run(){
224      def sql = Sql.newInstance(DB,USER,PASSWORD,DRIVER)
225      while(true){
226        def MuseumData request = serverRequest.read()
227        println("DB: Received serverRequest")
228        def location = request.getLocation()
229        if(request.getRequestType() == 1){
230          def answer = 0
231          sql.eachRow("select * from museumitem mi, museumareapoint map
232                       where map.address =? and mi.active =? And
233                       map.active=?",[location,"1","1"]){ row -> println
234                       "${row.name}"
235                       answer ++
236                       }
237          if(answer == 1){ println("DB: Only one item in this area point")…
238          }else if (answer > 1){
239            println("DB:more than one item: ${answer} in this access
240  point")…
241          }else{
242            //no items available in this location so send a message back to
243            //the user
244          }
245        }else if(request.getRequestType() == 2){
246          println("DB: Single item request")
247        }
248        //now send the request object back to the museum server
249        println("DB: Sending request (MuseumData) to client")
250        sendItemDetails.write(request)
251      }
252    }
253  }
```

Listing 19: Database Server Process

Obviously if a user has already selected to view a single item this query is bypassed {245} by reading the information stored in the `requestType` variable that is stored within the `MuseumData` {226, 229} object that is always passed during communication between the `client` and `server`.

An additional task completed by the database server is the communication to the Image and Sound Servers. The main server process communicates with the database server which then delegates responsibility to the image and sound servers when required. The database server then accumulates all the data from the processes and wraps it in a `MuseumData` object before sending it back to the `server` {250} which then passes it to the `client`.

4.2.6 Sound

The Sound within the application takes a similar approach to the Image process as described in Section 4.2.4. Using the same serializable object, `ByteBuffer`, the Sound process on the server side uses an additional package called FreeTTS as shown in Figure 29.

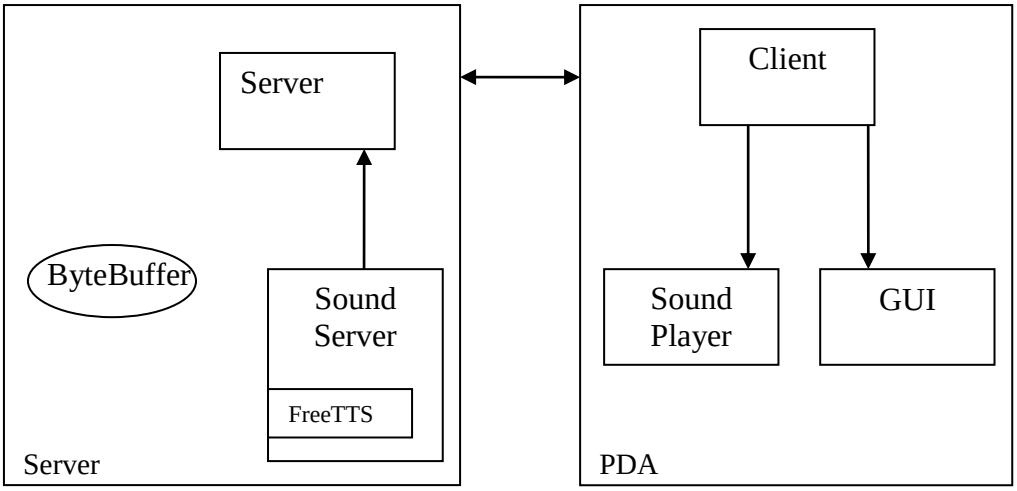


Figure 29: Application Architecture with the additional sound processes

Listing 20 is an example of part of the `SoundServer` process showing, if required, the creation of a .wav file. The second part of the `SoundServer` is shown in Listing 21 where the .wav file is read and converted into a byte[].

```
254  class SoundServer implements CSProcess {
255    def ChannelInput soundRequest
256    def ChannelOutput sendSound
257    void run(){
258      while(true){
259        def MuseumData md = soundRequest.read()
260        if(md.getRequestType() == 1){
261          def text = md.getMessage()
262          VoiceManager voiceManager = VoiceManager.getInstance()
263          Voice voice = voiceManager.getVoice("kevin16")
264          if (voice == null) {…}
265            voice.allocate()
266            AudioPlayer ap = new SingleFileAudioPlayer("${location}
267                                ${md.getLocation()}",
268                                AudioFileFormat.Type.WAVE)
269            ap.setAudioFormat(new AudioFormat(8000, 16, 1, false, true))
270            voice.setAudioPlayer(ap)
271            voice.speak(text)
272            voice.deallocate()
273            ap.close()
274          }
```

Listing 20: SoundServer process using FreeTTS

The `AudioPlayer` {266} works with byte arrays to play sound depending on the current `AudioFormat` {268,269}. The `SingleFileAudioPlayer` {266} writes the data to a sound file instead of the `AudioPlayer` playing the sound in real time. One of the

parameters required is the type of file format which is `WAVE` {268}. The `setAudioFormat` {269} will format the data in terms of the sample rate, number of bits per a sample and the number of channels to be used. The `Voice` class is responsible for translating the ASCII text into speech using a lexicon that matches the vocabulary into spoken words {263, 271, 272}. FreeTTS has some basic lexicons that use computerised speech which is weak in quality and counter intuitive but acceptable as part of this prototype. FreeTTS does offer extensions where other voices can be recorded and easily implemented within the system so it has the same functions as the basic version. The `VoiceManager` {262} provides an interface to the available voices {263}. Once the .wav file has been created the voice uses the `AudioPlayer` to write the spoken version of the text to the file using the method `voice` {266, 273}. Once written, the `voice` and `AudioPlayer` are closed and saved {272, 273}.

```
275    ByteBuffer bb = new ByteBuffer()
276    bb.type = 2
277    try{
278      InputStream is = this.getClass().getResourceAsStream("sound/" +
279                       "${md.getLocation()}" + ".wav")
280      if (is.available() != null) {
281        BufferedInputStream bis = new BufferedInputStream(is)
282        ByteArrayOutputStream baos = new ByteArrayOutputStream()
283        int ch
284        while ((ch = bis.read()) != -1) {
285          baos.write(ch)
286        }
287        bb.b = baos.toByteArray()
288      }
289    }catch(Exception e){bb.type = -1}
290    sendSound.write(bb)
291  }
```

Listing 21: SoundServer process converting a sound file to a byte[]

The translation of the sound file to a `ByteBuffer` is very similar to the image process as the file is read from the file server and then translated into a byte [] {278 - 287}. The difference is the type assigned to the `ByteBuffer` as it has the value 2 instead of 1 {276}. This information is used in the client process to determine the delegation protocol for the `ByteBuffer` object, i.e. the `AccessClientImage` or `AccessClientSound` process. These processes will translate the `byte[]` into the correct format.

`Sun.audio` package provides a useful class that takes a `byte[]` and plays the sound in a single utterance. The Java applet package provides a similar mechanism to this but is only available as part of the J2ME API.

```
292  public class AccessClientSound implements CSProcess {
293    private AltingChannelInput fromACC;
294    …
295    private void sunSound(byte[] b){
296      try {
297        InputStream in = new ByteArrayInputStream(b);
298        AudioStream as = new AudioStream(in);
299        AudioPlayer.player.start(as);
300      } catch (Exception e) {…}
301    }
302    public void run() {
303      while(true){
304        ByteBuffer bb = (ByteBuffer)fromACC.read();
305        sunSound(bb.getB());
306      }
307    }
308  }
```

Listing 22: AccessClientSound process

Listing 22 is an example of the `Sound` process on the client side that takes the `ByteBuffer` from the main `client` process {304} and passes the `byte[]` to the method `sunSound` {305}. This method creates a `ByteArrayInputStream` {297} and wraps an `AudioStream` {298} around it so the `byte[]` is treated as a sound object. The `sun.audio.AudioPlayer` takes the stream and plays it using the PDA's default sound settings {299}. The `run` statement is nested with a `while` loop so the user can replay the sound clip by pressing a replay button on the GUI {303}.

4.2.7 Final Diagram

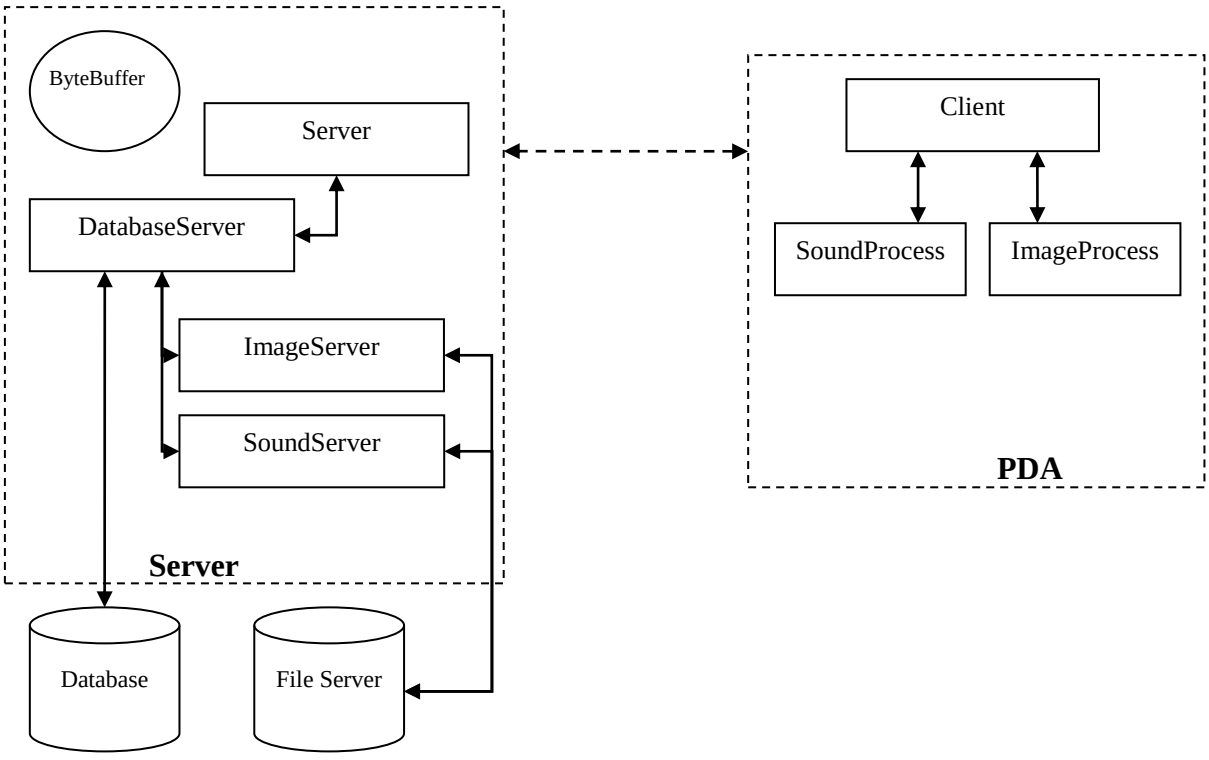


Figure 30: Overall Architecture of Visitor Application

Figure 30 shows all of the above implementations combined together into one diagram. It has however avoided the detail of all the processes on the client side except for the additional ones that have been created particularly for the museum application including the sound and image processes.

An additional item added here is the creation of a file server which will store the image and sound files. These will be retrieved by the Image and Sound Servers respectively. The next Chapter will detail the implementation of the Grails application and its role in managing the data that is stored in the database and file servers.

5 Grails

This chapter will discuss the implementation of the Grails application for the museum staff. Grails, as mentioned in Section 2.7, is a dynamic groovy-based web framework that removes the complexity of developing enhanced agile applications. During the design phase the concepts of GORM and domain classes were discussed. Figure 11 shows the framework's structure where Hibernate and Groovy relate to these concepts respectively.

5.1 Grails Structure

"Convention over Configuration" (Rocher, 2006a) is Grails advantage over other web based languages such as Coldfusion and PHP as it creates the outline of the system's structure when you generate the application. This enables convention as users will know where the new components are added and what the standard is for naming components so it matches the rest of the structure. Unlike the other web languages, new users can learn the Grail's framework and know that every project will adhere to the rules. The other web languages will have different concepts and structures that vary from developer to developer.

Once Grails has been downloaded and installed on the root directory, i.e. c drive, creating a project is relatively straightforward. Using a command editor, such as command prompt on Windows located within the required project directory, typing `grails create-app` will prompt a project name and result in the folder structure shown in Figure 31.

The folders that are highlighted by a rectangular box are the ones that have been used during the development of the museum prototype. `FreeTTS` is an additional folder used for the creation of Sound files discussed in Chapter 4 and not part of the grails framework. The `src` folder contains the visitor section of the application and the database

and is an area to store all java and groovy files that are not specifically related to the grails functionality.

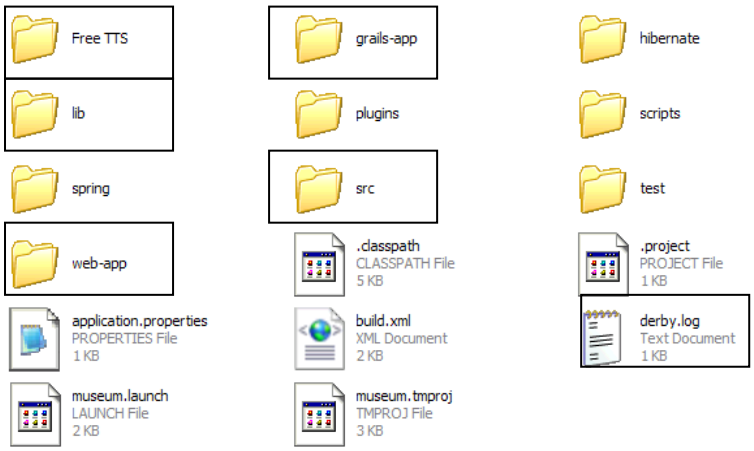


Figure 31: Grails Folder Structure

The `web-app` folder replaced the index.htm file, the default homepage for most websites, so the main page is the user login form. The `grail-app` folder is the most important folder in the grails application and Figure 32 shows the contents of this folder.

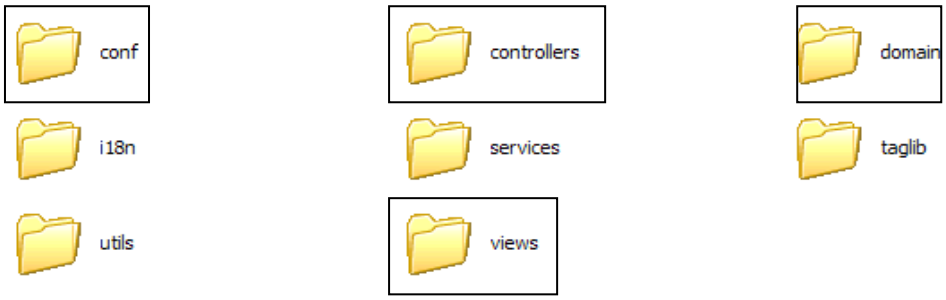


Figure 32: Grails application folder structure

Once again the boxes represent folders that were accessed and modified for the museum prototype. The `domain` folder contains all the domain-classes written in Groovy. Returning to the convention element, Grails understands that any domain class contained within this folder should offer persistence so will automatically create an entry within the underlying database. The `conf` folder contains data source details for different environments, i.e. development, test and live, so this will need to include the necessary configuration setup for Apache Derby as outlined by Tang (2007). The `views` folder contains Groovy Servlet Pages (.gsp) files that are similar to .php or .cfm files where HTML is mixed with the dynamic web language. This will be discussed in detail in

Section 5.4. The `controllers` folder contains a file for each domain-class and is the final piece of the main characteristic of grails; a Model – View – Controller architecture.

5.2 Model

Figure 32 shows the main concept of grails, minus the `conf` folder, which is a MVC. The domain is the model that not only offers persistence for database compatibility but also outlines the main functionality for that part of the application. Listing 23 is the domain class for the `Museumareapoint` written in Groovy. During the design, four main classes were identified to implement the main functionality of the system; Users, Museumarea, Museumareapoints and Museumitems. Therefore each of these has been implemented as a domain class.

```
309  class Museumareapoint {
310   Museumarea ma
311   String name
312   boolean active
313   String address
314   static belongsTo = Museumarea
315   static hasMany = [museumitems: Museumitem]
316   static constraints = {
317         name(maxLength:100, blank:false)
318         address(maxLength:15, blank:false)
319         active()
320         ma(blank:false)
321   }
322   String toString() {"${ma.name} : ${this.name} : ${this.address}"}
323  }
```

Listing 23: Museumareapoint domain class

The `Museumareapoint` class in Listing 23 has 3 attributes {311, 312, 313} and shows the different types of relationships available within the Grails framework; `belongsTo` {314} and `hasMany` {315}. Reflecting the class diagram in Figure 20 the `Museumareapoint` is associated with a `Museumarea` {314} but also contains a collection of `Museumitems` {315}. The `static constraints` {316 - 321} is a form of validation that will be used during the database setup for that table and also determine nullable and size restrictions. The `toString` method {322} will be used for displaying the item on a webpage.

5.3 Controller

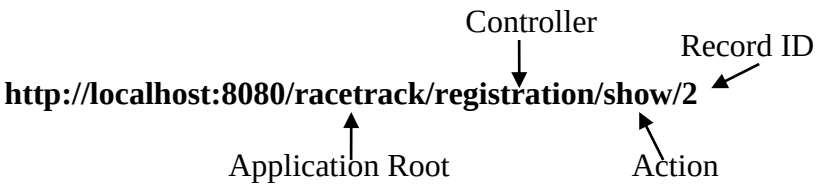


Figure 33: Grails URL (adapted from Rudolph 2006, Figure 3.3)

Grails can be adapted to have as much or as little functionality as required. It will initially create the fundamentals but offer ease of extending functionality. As previously mentioned, a domain class has a controller and a view to manage the actions and display options respectively. The command `grails generate-all` will create an extensive selection of views including an `edit`, `create` and `show` page. The controller will be similar to the one shown in Listing 24 that has a function for each type of page that is available. Figure 33 shows the structure of a URL in Grails and how the different entries map to the underlying folder structure. Using the above Figure, racetrack is the name of the application or project and registration is the name of a domain class and the corresponding controller. The next parameter 'show' is an entry expected in the controller class, similar to the `list` {328 - 331}, `show` {332 - 334} and `delete` {335 - 345} entries in Listing 24. The last parameter is an id that can be used within the controller. In Listing 24 the `show` function uses the `id` to retrieve the correct `museumareapoint` {333}.

```
324  class MuseumareapointController extends BaseController {
325    def index = { redirect(action:list,params:params) }
326    def beforeInterceptor = [action:this.&auth]
327    …
328    def list = {
329      if(!params.max)params.max = 10
330      [ museumareapointList: Museumareapoint.list( params ) ]
331    }
332    def show = {
333      [ museumareapoint : Museumareapoint.get( params.id ) ]
334    }
335    def delete = {
336      def museumareapoint = Museumareapoint.get( params.id )
337      if(museumareapoint) {
338        museumareapoint.delete()
339        flash.message = "Museumareapoint ${params.id} deleted."
340        redirect(action:list)
341      }else {
342      flash.message = "Museumareapoint not found with id ${params.id}"
343      redirect(action:list)
344      }
345    }
346   …
347  }
```

Listing 24: Museumareapoint Controller

5.4 View

The view component is similar to a plain HTML page with added Cascading Style Sheet (CSS) entries and additional functionality called Tag Libraries (Rocher, 2006b). Listing 25 is an example of the `show.gsp` page whose graphical representation is shown in Figure 34. Some of the key functionality is shown in the Listing including `<g:render … />` {349} which calls the template `menubar` and displays the information within that area of

the page. This is useful for navigation bars, headers and footers that are consistent on all pages.

```
348  <html>…
349    <g:render template="/menubar" />
350      <div class="nav">
351        <span class="menuButton">
352          <g:link action="create">New Museumareapoint</g:link>
353        </span>
354      </div>
355      <div class="body">
356        <h1>Show Museumareapoint</h1>
357        <g:if test="${flash.message}">
358          <div class="message">${flash.message}</div>
359        </g:if>…
360          <tr class="prop">
361            <td valign="top" class="name">Id:</td>
362            <td valign="top" class="value">${museumareapoint.id}</td>
363          </tr>…
364          <tr class="prop">
365            <td valign="top" class="name">Ma:</td>
366            <td valign="top" class="value">
367              <g:link controller="museumarea" action="show"
368                 id="${museumareapoint?.ma?.id}">${museumareapoint?.ma}
369              </g:link>
370            </td>
371          </tr>
372          <tr class="prop">
373            <td valign="top" class="name">Museumitems:</td>
374            <td  valign="top" style="text-align:left;" class="value">
375              <ul>
376                <g:each var="m" in="${museumareapoint.museumitems}">
377                  <li>
378                    <g:link controller="museumitem" action="show"
379                       id="${m.id}">${m}
380                    </g:link>
381                  </li>
382                </g:each>
383              </ul>
384            </td>
385          </tr>…
386          <div class="buttons">
387            <g:form controller="museumareapoint">
388             <input type="hidden" name="id" value="${museumareapoint?.id}"/>
389             <span class="button"><g:actionSubmit value="Edit" /></span>
390             <span class="button"><g:actionSubmit value="Delete" /></span>
391            </g:form>
392          </div>…
```

Listing 25: A View page in Grails

The `<g:if` {357 - 359} is a grails IF statement that will display any error messages regarding data entry or validation. When the constraints, as shown in Listing 23 {316 - 321}, are applied to a domain class, grails will automatically update basic validation within the create or edit pages for that class. Any parameters passed from the controller are output using `${}` {376}. The `<g:each` {376} is similar to a groovy closure and will display each `museumitem` that is associated with this `museumareapoint.` It also includes a link `<g:link` {378 - 380} to their show page by calling the function `show` {378} within the `Museumitem` controller. The last grails tag is `<g:form` {387 - 391} which is similar to html but will pass any parameters stored on the page {388} to the

`Museumareapoint` controller. Depending on the button pressed this could be the `Edit` {389} or `Delete` {390} functions.

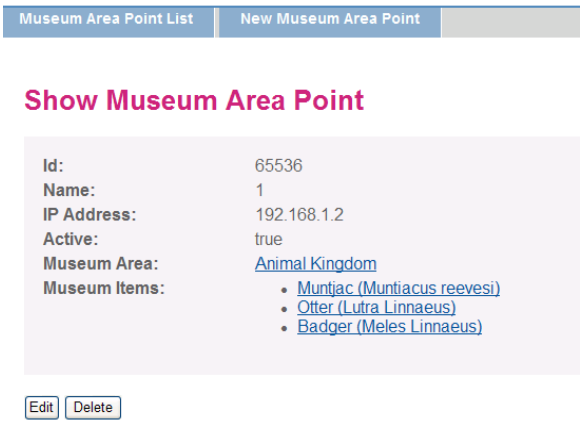


Figure 34: The View for the Museumareapoint 'show' function

5.5 Summary

The above implementation has focused on the museumareapoint class as shown in Figure 20. Grails uses "*convention over configuration*" so the implementation of the User, Museumarea and MuseumItem are similar to the MVC components described above. As a result the process of implementing these areas in the application will not be discussed to avoid repetition.

6 Testing

Testing is a crucial element of any successful project and helps to identify, at an early stage, any outstanding tasks, problems and modifications that are required to meet the overall objectives. This chapter will continue with the incremental development approach taken during the implementation stages. Each stage will be tested and carried forward to be integrated with the next stage.

6.1 JCSP Structure

The initial development outlaid the structure of the visitor application. One of the key functionalities required for the application was to communicate over a TCPIP network. Therefore the first increments of development created a client-server architecture. This was then applied to the concept of JCSPNet by running a CNS that both the client and server connected to initially to establish a connection with the network.

To elaborate on this prototype, the concept of JCSPMobile was then combined within this prototype so mobile processes could be downloaded to the PDA. Development within this area has been written and discussed in papers by Chalmers et al (2005) and Kerridge et al (2006) so testing this functionality was not required as many examples were already available.

6.2 Graphical User Interface

The GUI is the visitor's access to the application and therefore must provide interaction that returns or results in what the visitor will expect. The GUI uses buttons, a text area and a text entry field to enable the user to communicate with the application. Figure 35 is an example of the finished GUI highlighting the different components.

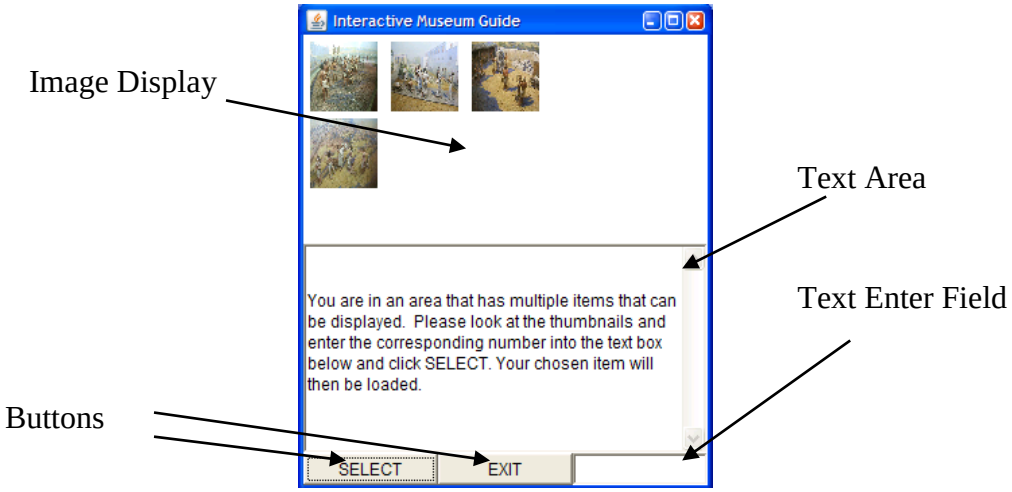


Figure 35: Graphical User Interface

Listing 26 is an example of some console output from the process listening to the GUI running on the PDA. Firstly the user selects START {393} to begin access to the museum application. The next step would be for the user to select a valid choice from a collection of thumbnails using the text enter field {396, 397}. This action would cause a refresh of the GUI so the thumbnails are replaced by a larger single image of the chosen item and the textarea updated to display information of the item. Once the user is finished then can press back or choose to exit the application {399, 400}.

```
393  Button Pressed: START
394  System is now ready to continue to display an item or thumbnail
395  Updated Console Message
396  Selection 1
397  Valid selection
398  System is now going to return to previous screen
399  Updated Console Message
400  EXIT Pressed
```

Listing 26: Testing of GUI

6.3 Image

The Image was divided into smaller sub tasks to enable the implementation of this functionality to be manageable. The first stage of the implementation was to display an image on the client side without any conversion. The task was completed via the simulator on a single laptop so both client and server could access the same files. This test used a simple GIF image and the components that relate to the ActiveCanvas, as discussed in Chapter 4, and was successfully completed.

The following stage resulted in the conversion of the image into a byte array. Again this was implemented on the client process using the simulation environment. The Image was

located on the file server and read in as a stream. The stream was then converted to a byte array so it could be tested to determine if it could successfully create an Image object. This functionality caused difficulties as the Java Standard Edition (J2SE) has basic protocols for converting a byte array into an Image object. This resulted in lots of testing and research using the Java Graphics2D API (Sun Microsystems, 2007a) and alternative J2ME classes that are enabled in Applet classes. Eventually the option of the Toolkit package became apparent (Zukowski, 2000) and is shown in Listing 18 on line 193. This test proved that it was possible to convert an image, read from the server, into a byte array and then effectively be recreated back to an Image object using the Toolkit package. The next step was to test this functionality on the server side so the byte array could be wrapped within a ByteBuffer class and then sent over the network. It was at this stage that the type byte array became an issue for the JCSPMobile classes.

A class called DeserializationFilter.java within the JCSPMobile package disliked the type 'byte[]' and as the source code was not available for the JCSPMobile package the solution was to contact the developer, Kevin Chalmers. It seemed that the dynamic class loader concept previously described in Section 2.4.3 was trying to download the class for a byte array but was failing to get an efficient copy. After several iterations it was summarised that a bug in JCSPMobile had been found and that the expected hierarchical dynamic class loader inheritance was not being applied. The class dealing with the byte array was not calling its parent class to help retrieve the classes for a byte array and therefore terminated the application. Chalmers revised the JCSPMobile package and created a new version that added essential code so the hierarchical search was applied to all dynamic class loading. As a result the ByteBuffer was wrapped in a MuseumData class so it could be transmitted over the network and decoded effectively by the client process and therefore using the previous tests, displayed images on the Active Canvas of the GUI.

6.4 Database

Apache Derby provides some useful documentation and tutorials via its Eclipse IDE plug-in. At this stage the Grails application had not been considered as an additional system for the project so all of the database tables were designed and populated using the ij console that came with the Eclipse plug-in. Barclay and Savage (2007, Chapter 17) provides an essential chapter on the concept of SQL querying in Groovy and therefore resulted in this implementation to incur no difficulties or problems. When tested the

Database process on the Server side returned the expected results from the database and, when passed a parameter, filtered the records accordingly to return the expected response. The application was then modified to have the Database process to control the request of images so additional channels provided the mechanism to request and receive data from the Image server. The Image Server waited for a Record ID from the database that matched a corresponding GIF located on the file server. As mentioned in Section 4.2.4 the image process returns a ByteBuffer so the database process obtained the ByteBuffers and arranged them in an array that was stored as a variable within the MuseumData class. The Database then returned the result to the Museum process who forwarded it to the client. This integration caused no problems or affects for the application and the functionality was easily implemented and tested.

Client Process	Database Server Process	Image Server Process
Setting request type to 1		
	Received serverRequest The request is 1 and location 127.0.0.1	
		images\ Image request 196611 21686 Setting ByteArray Setting ByteBuffer type to 1
	Received Image	Sending ByteBuffer Waiting for next image Image request 196612 20480 Setting ByteArray Setting ByteBuffer type to 1
	Received Image … x 2 more times Image size: 4 Sending request (MuseumData) to client	Sending ByteBuffer … x 2 more times waiting
Received Museum Data Thumbnails received Al Size 4 amount of images Creating Graphics Command Looping through item 1 Image ID 196611 Processing Image Looping through item 2 Image ID 196612 Processing Image … x 2 more times	sent data Waiting	

Figure 36: Retrieving Item details from the server processes

Figure 36 shows the three processes; Client, Database Server and Image Server. Time flows downwards with entries in the column representing the current action taking place in that process. It can be seen that the client sends a request to the server where the database retrieves the request type and location, i.e. IP address. The Database Server then queries the database to find out how many items reside in this museum area and requests

the thumbnail images from the Image Server. Once complete the Database Server gathers the thumbnails and creates a MuseumData object that is then sent back to the client. The Client then recreates the images and adds them to the GraphicsCommand waiting to be displayed on the ActiveCanvas.

6.5 Sound

Testing of sound was initially applied by simulating the application on a single computer. FreeTTS provides useful demos about how to take in some ASCII text and apply one of their default voices to create spoken vocabulary. The requirement of saving the spoken vocabulary into a wave file was more challenging but, once applied, successful. The mechanism of converting the sound file to a byte array had the same process as the Image process and was therefore easy to apply. Once the Sound properties had been wrapped in a ByteBuffer the transfer over the network was nondescript and arrived as expected with the client. A separate process dealt with the sound properties to convert the byte array so it could be played using the default sound properties on the PDA.

This test worked fine using the sun.audio package that is not publicised but available without documentation. The classes within the package are used by Sun for personal use within their company but are available to be applied by the public. During simulation testing this mechanism of playing sound, as shown in Listing 22, worked successfully. However once this test was applied to the PDA an error message occurred regarding Serialisation ID incompatibility. Further research resulted in the implementation shown in Listing 27 which is taken from the documentation of the Java Sound API (Sun Microsystems, 2007b).

There are three things that are required for sound to be played within a J2SE Java program using the Sound API; audio data, mixer and a data line. The audio data is the byte array that originated from a valid sound file, i.e. wave format. The mixer manages the different streams of audio so they are combined into one stream that can then be interfaced to the devices main audio output controls. The data line manages the playing of the sound by setting the sound's characteristics such as the number of channels (mono or stereo), the sample rate, frame size and the number of bits per a sample. These attributes can then inform the mixer how to apply the sound file so the output is correctly configured.

```
401   private void javaSound(byte[] b){
402     InputStream input = new ByteArrayInputStream(b);
403     AudioInputStream ais = AudioSystem.getAudioInputStream(input);
404     final AudioFormat format = ais.getFormat();
405     DataLine.Info info = new DataLine.Info(SourceDataLine.class, format);
406     SourceDataLine line;
407     try {
408       line = (SourceDataLine)AudioSystem.getLine(info);
409       line.open(format);
410       line.start();
411       } catch (LineUnavailableException e) {…}
412     int bufferSize = (int) format.getSampleRate() * format.getFrameSize();
413     byte buffer[] = new byte[bufferSize];
414     int count;
415     try {
416       while ((count = ais.read(
417         buffer, 0, buffer.length)) != -1) {
418         if (count > 0) {
419           line.write(buffer, 0, count);
420         }
421       }
422     } catch (IOException e) {…}
423     line.drain();
424     line.close();
425   }
```

Listing 27: New Sound Method

Listing 27 is the revised sound playing method using the Sound API. Firstly the byte array is read {402} and converted it to an `AudioInputStream` {403}. This ensures that the sound's characteristics are obtained as details of its channels, sample rate and frame size are determined by the FreeTTS package and unknown to the application. The `AudioSystem` {403} interrogates the device for its current sound settings so it knows what settings should be applied to successfully play the sound. A `SourceDataLine` 406, 408} is created to set up the streaming of the sound to the device's mixer and will manage the playing and stopping of the audio {410, 424}.

This new version worked successfully using the simulator but when tested on the PDA caused errors regarding the unknown type of SourceDataLine and therefore, after several tests, concluded that the compiled version of the Java classes of JCSPMobile on the PDA is different to the compiled versions of the museum application. Therefore without a copy of JCSPMobile, so it could be compiled with the museum application so the settings matched and then update the PDA's versions of the software, the sound on the application cannot work. After discussions with Chalmers he suggested compiling the museum application with the Java version 1.3 using the NetBeans IDE with the additional mobility plug-in. However the NetBeans IDE only has the Groovy compiler available within their 6.0 developer edition which is a Beta version and in its current state has a bug when downloading the Groovy compiler. Therefore this cannot be implemented until this bug

is resolved but it is feasible, once the bug is fixed, to have the sound converted, transferred and played successfully on the client's device.

As this problem became apparent the Java Media Framework (JMF) was discovered as an additional solution. However research concluded that the sound playing capabilities are only available if the JMF package is available on the mobile device (Colombo, 2002). This it is not a requirement for any mobile device manufacturer and therefore an unfeasible solution.

6.6 Grails

Grails offers, as part of its framework, the ability to apply unit and functional testing but as it was an additional objective has not been covered in this project. Instead its high level implementations enable the developer to quickly create the necessary objects required which are then reflected in the database using Grails GORM concept. Each of the three objects (area, access points and items) were easily constructed using the Grails framework and using the command `grails generate-all` creates a default look of pages and controller entries so adding, removing and editing objects are easily managed. As this is a default implementation of Grails, testing was not a major concern.

However the ability to add images for a museum item became an issue as the Grails default configuration caused problems. The pages for editing and adding items use simple html such as input, text areas and drop down fields. When the form is submitted the objects controller obtains the parameters using the `params` attribute to obtain all of the the `key:value` entries. However when you add the option of file uploads the data encryption type is changed that the `params` attribute no longer works and therefore a loss of entries. To overcome this issue an additional link was added to the Museum Items list page that had a hyperlink "`Add Image`". This mechanism enabled images to be saved and renamed so the visitor application could successfully display thumbnail and images that correctly correspond with the selected item as shown in Figure 35 on the GUI of the visitor application.

7 Usability

This chapter will revisit the PACT analysis framework described by Benyon et al (2005) and the issues highlighted in Chapter 3.

7.1 People

Chapter 3 highlighted some People issues that would need to be considered during the implementation of the system. In particular a key element was to provide the information in various formats to ensure that different users and their abilities would be able to use the application.

The prototype currently displays text and images and plays sound, on the simulation, that all represent the same theme. The images are exact representations of the items shown in the museum and the text and sound are replicates to enable different abilities such as hearing impairment to receive the same information as someone that is not. The system has restricted the graphical user interface to five basic components as shown in Figure 35, the GUI of the visitor application.

The People principle of PACT also identifies the safety and ethical issues. Safety is a crucial attribute in any system development and especially when human-centred. Sound has been added to the application to prevent visitors reading the description whilst walking around the museum. The GUI is simple to assist with the safety aspect of the system and also to promote positive ethics.

7.2 Activities

The main activity of the system is to allow visitors to use their own mobile device to view museum items. As part of the Activities principle, five subsections have been devised as shown in Section 3.1.3 to thoroughly understand what is expected of the system.

7.2.1 Temporal

The regularity of a task needs to be considered when designing a system and with it the necessity of documentation. The prototype has one main task with three accompanying screens. Depending on the visitor this task may be completed several times in one visit or once. As it is a regular task, documentation can be maintained in the start screen of the application. If preferred a simple leaflet can be available for visitors requiring further guidelines and documentation.

Temporal also considers the impact of interruptions for the system and user. The system has maintained a simple GUI and easy navigation. Therefore visitors should be able to replay items that they have missed or misunderstood. JCSPMobile enables the system to be dynamically downloaded to the mobile device and therefore prevents corruption if there was a system failure.

7.2.2 Cooperation

As highlighted in Section 3.1.3, the project cannot assume that everyone will have their own device. However a family may have access to one or more devices and therefore the ability to share as a group is important. There is no significant difference to the application being used as a group or as an individual but the application will replay sound snippets when prompted.

7.2.3 Complexity

The prototype represents one sequential task with three main screens; Welcome, Thumbnail display, Item View. Once the visitor passes the Welcome screen they will enter the main thumbnail view as shown in Figure 37. The user can then navigate between the two different screens as often as they like

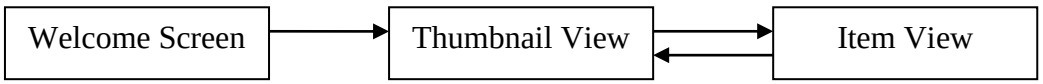


Figure 37: Workflow of visitor application

Figure 35 shows the main GUI of the application with two buttons in the lower left hand corner as the main navigational components. The buttons will always provide the user with the option to proceed or return to the previous screen.

7.2.4 Safety Critical

Safety is not always about accidents and injuries, although they are considered in this section, but can include the recuperation of a system if the user makes a mistake. The current prototype is basic with little accommodation for confusion. If a user clicks the wrong button or selects the wrong thumbnail, the navigation protocol is easy for the user to return to the previous screen. If the user exits from the application then they can easily restart and connect to the CNS again. The prototype should be an enhancement to a visitor's experience and not a hindrance which includes the risk of safety. The prototype cannot distract the user or disturb others from their visit therefore the design was considered carefully. The three different options of media was a key decision, providing replicated material prevents the visitor looking at the mobile device for prolonged periods of time.

7.2.5 Nature of Content

JCSPMobile enables processes to be downloaded to a visitor's mobile device without a requirement for software to be saved in memory. A prerequisite is a simple java class on the mobile device that is running and connected to the CNS. The current prototype works with a PDA with a stylus and touch screen for data entry. Mobile phones without touch screens will require a modification to the application so that simple keypad entry is enabled.

It is advantageous for the device to have a colour screen monitor but the prototype will work for both scenarios. The device must have the JVM loaded and if required sound output but similar to the screen is not necessary.

7.3 Context

Context considered the design in terms of the physical environment within three subsections; Physical, Social and Organisation.

7.3.1 Physical

Physical considers the museum environment and how the system will work and be used in the different buildings. Unfortunately the system has not been tested within the museum environment. Keys issues that would need to be observed when tested are the sound quality and the impact of the clarity of screen in different lightings.

7.3.2 Social

Social was discussed in Section 6.2.2 and highlighted that there are no significant differences to the system when used by an individual or as a group. Testing has been applied to an individual but not as a group activity.

7.3.3 Organisation

Applying new functionality within the museum will affect the staff and this is what the organisation context considers. The additional Grails application developed for the museum staff should improve the impact of the new system. Keeping with the same principles of the visitor application, the grails application contains a small number of screens to prevent uncertainty and therefore assist the staff in adding and editing items.

8 Bluetooth

Chapter 2 discussed mobile devices and their available communication protocols. One that was mentioned which is unique compared to the others was Bluetooth. WiFi has been used for the museum prototype but the environment that was discussed in Chapter 3 identified that a number of artefacts can be displayed within a confined space. Therefore the large range of WiFi is not ideal for the museum application and the possibility of implementing Bluetooth would be beneficial.

8.1 Characteristics

Bluetooth uses the 2.4 – 2.4835 GHz ISM (Industrial, Scientific, Medical) radio band. This band is available globally and used by other wireless protocols. To overcome this, Bluetooth uses the concept of channel hopping so it changes the dedicated channel frequently. The frequency is divided into 79 different channels with 1 MHz intervals and changes by1600 hops/sec. The devices create an ad-hoc Piconet or Personal Area Network (PAN) which are mini networks to enable up to seven different Bluetooth devices to communicate and remain connected during channel hopping. The scheduling technique is round robin so all devices get an equal share of the network.

Bluetooth comes in different classes that relate to their distance capacity so mobile devices are a class 2 Bluetooth that enables a 10 metre range. This restriction in distance enables Bluetooth to be suitable for battery powered devices as it results in low power consumptions of 2.5MW.

8.2 Architecture

Figure 38 shows the protocol stack that all Bluetooth enabled devices must implement. JCSPMobile is currently developed for the WiFi communication protocol and would therefore need to implement the main elements of the protocol stack.

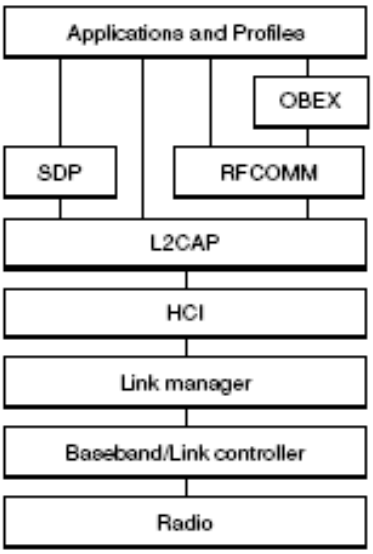


Figure 38: Bluetooth Protocol Stack (Apple Inc, 2007)

The Radio layer describes the characteristics of the radio frequency signals for sending and receiving. The next three layers manage and format the communication of links. There are two different types of links used within Bluetooth with corresponding packet types used for data transmission (Apple Inc., 2007).

- *Synchronous Connection-Orientated (SCO),* for isochronous and voice communication, an example is headsets. A channel is reserved for periodic communication between the devices.
- *Asynchronous Connectionless (ACL)* for data communication, an example is the exchange of vCards (an electronic business card). A connection is established immediately between the two devices and packets are split into small segments.

The Host Controller Interface (HCI) layer is the boundary between the lower physical layers of Bluetooth and the remaining high level layers. The upper layers are determined by the sort of device being implemented. Desktop computers might implement their own layers but a device such as a headset may combine the layers so it remains compact. The HCI determines the protocol to be used. The upper layers are responsible for establishing connections, multiplexing between different applications and translating data into the correct format for the lower levels. Bluetooth devices can offer services if they are a server role.

The Service Discovery Protocol (SDP) outlines the actions available as a client or server device. The L2CAP has a reserved channel used for a client-server connection that a

client can connect to a server device that will list its available services that are stored in a local Database. The client can then establish a dedicated connection to this service.

8.3 Profiles

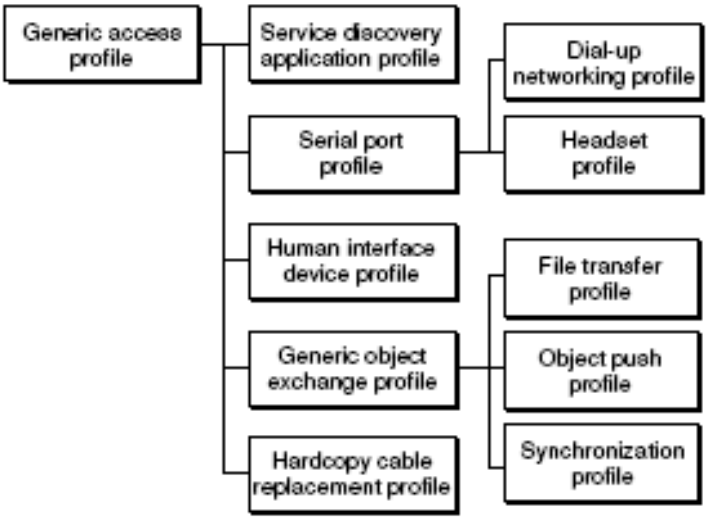


Figure 39: Bluetooth Profiles (Apple Inc., 2007)

Figure 39 lists the different types of profiles identified by the Bluetooth Special Interest Group (SIG). The profiles provide guidelines and support to help development of applications within that specified area so it can be integrated successfully with Bluetooth.

The Generic Access Profile (GAP) provides the basic Bluetooth functionality that all other profiles implement. The other profiles are used in the following areas as described by Apple Inc. (2007):

- *Service Discovery Application Profile* provides guidelines at how to use the SDP layer to search for available services within the PAN.
- *Serial Port Profile* enables legacy systems to treat a Bluetooth connection like a serial-cable connection.
    - *Dial-up Networking (DUN)* provides the mechanism for a laptop computer to use Bluetooth to connect to a mobile device so it can use their telephone-based network.
    - *Headset Profile* describes how to implement an additional device to be configured as a mobile phone or laptop's main input/output interface and as the name suggests is the technology used by a Bluetooth headset.
- *Human Interface Device (HID)* implements a Bluetooth link using USB devices

- *Generic Object Exchange Profile* uses the OBEX layer, shown in the protocol stack, to enable a client-server relationship for object transmission.
    - *Object Push Profile* uses a particular object format that is used by the VCard for successful transmission of information.
    - *File Transfer Profile* provides guidelines to help successful transmission of files and folders that are not of the format of the VCard.
    - *Synchronization Profile* provides guidelines to help devices synchronise information such as a PDA and laptop sharing the same address book.
- *Hardcopy Cable Replacement Profile* relates to communication with a printer so rendered data can be sent via a Bluetooth connection.

8.4 Recommendation

This Chapter has discussed Bluetooth and its structure. It also identified a number of profiles that the Bluetooth SIG created to help provide guidelines for common application types that want to integrate with the Bluetooth technology.

In terms of the Museum Application, the underlying technology JCSPMobile will need to add an implementation for Bluetooth within its communication section that adheres to the guidelines outlined by the Bluetooth SIG.

9 Conclusion

This Chapter will revisit the objectives identified in Section 1.3 and evaluate if the current museum application meets their expected requirement. Later the Chapter will discuss further work and in particular how the technology used within the museum application can be applied to other sectors. The last section of this chapter will evaluate the overall project process.

9.1 Summary

Six objectives were outlined at the beginning of the project and were aimed to be achieved by the museum prototype. Individually these objectives will be discussed and evaluated, in order, to determine if they have been met.

Objective 1. *The project intends to use Mobile Devices to interact within a museum environment. The first stage is to create a program that downloads effectively to a Mobile Device without being disruptive or unfriendly to the visitor.*

Section 4.2.2 discussed the implementation of JCSPMobile which enables mobile processes to be downloaded over a network to a mobile device. Therefore this objective has been achieved and successfully tested as discussed in Section 6.1.

Objective 2. *Once the basic interactions are working, building on this to include images that relate to the current item that the visitor is currently viewing.*

Once again revisiting Chapter 4, the implementation took an incremented approach as the objective suggests and therefore Section 4.2.4 discusses the Image functionality. This was tested in Section 6.3 where the downloading of different images and reading user's input was successfully achieved. The Image functionality was also revisited in Chapter 7

when the user evaluation was discussed and identified that users could successfully select an item from a collection of thumbnails. This resulted in a larger image of the chosen item. However it must be added that initial testing using the byte array concept did cause problems during data transfer and resulted in a new version of JCSPMobile being implemented. Therefore this objective was successfully achieved by the museum application.

Objective 3. *Mobile Devices are compact and therefore images will be small and, as a consequence, lack detail. Text will be short and precise so adding sound can represent a detailed description of the item and will add an additional feature to improve the interaction for the user.*

During testing a problem became apparent with this objective due to the current state of the technology. The initial implementation discussed in Section 4.2.6 used the sun.audio package for playing sound files on the PDA. The functionality worked fine during simulation on a laptop but when the process was downloaded to the PDA an error regarding an incompatible Serialisation ID became an issue. Therefore further work discussed in Section 6.5 worked fine during simulation but still resulted in errors when downloaded to a PDA. Therefore this technology is not available on the PDA but works successfully when simulated. The problem relates to different Java Runtime Environments (JREs) and compilers. The museum application compiles for Java 1.3 compatibility using the Eclipse IDE but the JCSPMobile package uses the same compiler but includes the mobility package available within the NetBeans IDE. However at the time of writing this report the development environment of NetBeans 6.0 has a known bug when trying to download the Groovy plug-in. Therefore the museum application could not be compiled successfully using NetBeans so the Serialisation IDs of the Java classes mismatch the software available on the PDA. Therefore this objective has been achieved using a simulation with the laptop but cannot be successfully downloaded to the PDA.

Objective 4. *Databases are useful for storing data of a similar nature and as the data being used by the program will include text, images and sound, it seems like a suitable storing device for the project. As mentioned the database*

will store details of the items within the museum and when a visitor is near a particular item, the details of that item are retrieved.

Chapters 4 and 5 discuss the use of the database for updating and retrieving information. Apache Derby was successfully used along with the Grails GORM concept to create a database that stored information of museum areas, access points and items. Grails offered an enhanced interface to manage the relationships between these three elements without the need to add additional tables to store foreign keys. Both applications successfully integrated with the database to update and insert information (Grails) and to retrieve information depending on the current location (visitor application). Additional to the database is a file server that stores the JPEG and WAV files for the images and sound respectively.

Objective 5. *A web based interface will enable the museum staff to update, when required, the details stored of the items within the Database (added during the course of the project).*

The Grails framework provided a successful application for integrating with the application database. It also offered a better interface than Apache Derby's standard ij console to manage the stored items. The security benefits were better as along with the username and password for the database the main page of the site is a login screen. A username and password is required, with appropriate validation, to access the site. Each element of the museum; areas, access points and items, have their own section within the site where museum staff can list, edit, add and delete records. Another positive feature of Grails is its use of Object Orientated concept so applying relationships between items was easily integrated. As shown in Chapter 5, museum access points can be easily associated with a particular museum area by using a drop down menu.

An additional page added to the site was the ability to upload images that are then renamed to match the Item ID and saved in an appropriate folder in the file server. Grails unfortunately does not offer the capabilities to add this functionality so it does not appear on the same page as adding a new museum item. This is due to the differences of form submissions as files use a different encryption compared to textfield entries.

Objective 6. Investigate Bluetooth as a potential communications medium for the museum environment.

This objective was changed during the course of the project as implementing the protocol within the time constraints did not seem a feasible option. Therefore Objective 6 was modified to a research objective to identify the work involved in implementing Bluetooth technology. Therefore Chapter 8 discusses the characteristics of Bluetooth and identifies its structure and various guidelines and procedures available. It however does not discuss what should be implemented within the museum application as the main communication protocol is established within the JCSPMobile package and out with the scope of the project.

9.2 Further Work

The project has demonstrated the potential for a commercial application within, in particular, the retail industry. Companies with loyalty schemes can encourage customers with offers and discounts using the mechanism used in the project for downloading processes. As customer details and spending habits are among the key information stored by loyalty schemes, the technology can use this data to personalise the experience for the shopper and hopefully increase revenue.

The simple mechanisms used in this project can be extended to include the ability to offer discount vouchers or advertising videos which can be downloaded to a customer's mobile device. The offers can then be stored in a central repository so any items bought by the customer using the virtual discount vouchers will automatically be taken off their total costs.

The key to successfully implementing this technology is to encourage the mobile industry to include the UASSSClient file on all devices and to enable wireless technology for commercial use. Another alternative is Bluetooth but due to the connection restrictions and limitations on retail floor space, this communication mechanism may be a backwards step.

Petrelli and Not (2004) identified that only 32% of visitors are new customers so there is potential in developing an application that is catered for returning users. As suggested returning visitors may want more information so developing a multi-levelled

customisable application would be beneficial. This application can ask the visitor about the detail they would like to receive about different museum items. Children and new visitors may opt for basic information while returning visitors may like snippets of curator documentation and therefore a more detailed specification of items.

The project used a PDA for testing the download of processes, however most mobile devices support the Java platform J2ME. Therefore the visitor application will need to be adapted so processes downloaded to the mobile device meet the requirements of J2ME which can vary from the J2SE implementations.

The application currently uses low resolution imagery but future developments may require better quality and larger sizes and therefore larger files to be transferred over the network. This requirement will limit performance but JCSP can implement the concept of buffering using a process that filters the writing of the image by interchanging between 2 or more channels. Therefore the image process will continually be receiving data but still remains the optimal solution for data transfer. Another option with images is been able to focus on particular items and their characteristics. The Graphics Command components enables many items to be displayed on the Active Canvas at once so and additional feature could be the creation of sensor points on the image highlighted, for example, with a rectangle. The user could click on the sensor points and obtain further information about the item.

9.3 Evaluation of the Project Process

The initial stage of the project was to identify the objectives for the application. The MoSCoW (Must have, Should have, Could have, Would like to have) acronym was used to divide the objectives into optional and essential requirements. The technique also helped to prioritise tasks as the first two (Must and Should) are regarded as the most important objectives.

MoSCoW	**Requirements**
Must Have:	JCSP, GUI, image, sound
Should Have:	JCSPMobile
Could Have	Bluetooth
Would Like To Have:	Imagery sensor points, better voice for sound

The Table above shows the objectives for the project using the MoSCoW technique.

Bluetooth was a Could Have requirement and therefore supports the change of the objective during project development.

The next stage took the above requirements and mapped them to a development plan using the concept of timeboxing. The timeboxing technique divides the total project duration and applies individual durations to the requirements relating to their priority and effort. The benefit of timeboxing is that it encourages development to be completed in small fixed time segments. The time segments enable the project to move from one section to another and concurrently complete areas. The project used timeboxing when developing an area of functionality. Testing for that functionality was completed straight away using part of the allocation of the testing budget. Then the write up for that area was completed. The alternative would be to finish all of the development, then test the prototype and finally complete the write up of the report.

On a personalise level, I found the timeboxing easier to remain focused on the overall project objectives and not on one area of functionality. JCSP was new technology for me so using smaller manageable sections, and incremental development, timeboxing helped to improve my motivation as there was realistic targets to achieve. A major disadvantage of the project process was the management of the plan. As time became sparse it was difficult to spend it updating the plan as it was more beneficial to spend the time with development or write up of the report. Therefore since completing this project, and studying project management, the benefits of a project manager have become clear.

The weekly Appraisal meetings were very useful and helped to keep the project within schedule as the Project Supervisor assigned weekly tasks that had to be achieved. The Poster Presentation provides an opportunity to reflect on the project and determine what has been achieved and, at a high level, how it can be shown.

The final benefit of the project was the opportunity to apply the theory learnt in previous modules during the course of the degree. Concepts such as UML, OOD, Java, Design Patterns and Groovy were taught in Software Development and Software Architecture modules. It is difficult to developing applications without considering the quality attributes learnt during the Software Quality module. Other modules that helped project development were Real Time Systems, which helped to understand the concurrency

concepts of Dijkstra and Hoare and Database Systems which helped to manage the creation and querying of the Apache Derby database.

References

Apache Software Foundation (2007, May 15). *Apache Derby*. Retrieved November 21, 2007 from http://db.apache.org/derby/

Apple Inc. (2007, November 12), *Bluetooth Device Access Guide*, Apple Inc., [accessed electronically]

Barclay, K. & Savage, J. (2007), *Groovy Programming: An Introduction for Java Developers*, Elsevier Inc.

Belapurkar, A. (2005, June 21), *CSP for Java Programmers: Part 1*, IBM, Retrieved July 24, 2007 from http://www-128.ibm.com/developerworks/java/library/j-csp1.html

Benyon, D., Turner, P. & Turner, S. (2005), *Designing Interactive Systems: People, Activities, Contexts, Technologies*, Addison-Wesley

Bluetooth (2007), *Bluetooth Basics*, Retrieved December 13, 2007 from http://www.bluetooth.com/Bluetooth/Learn/ http://www.bluetooth.com/Bluetooth/Learn/

Bluetooth SIG (2007), *Bluetooth Special Interest Group*, Retrieved December 16, 2007 from https://www.bluetooth.org/apps/content/

Bustard, D., Elder, J. & Welsh, J. (1988), *Concurrent Program Structures*, Prentice Hall

Chalmers, K. & Kerridge, J. (2005), *jcsp.mobile: A Package Enabling Mobile Processes and Channels*, Communicating Process Architectures 2005, IOS Press, 2005 [accessed electronically]

Chalmers, K., Kerridge, J. & Romdhani, I. (2007), *Mobility in JCSP: New Mobile Channel and Mobile Process Models*, Communicating Process Architectures 2007, IOS Press 2007

Chalmers, K., Kerridge, J. & Romdhani, I. (2006), *Performance Evaluation of JCSP Micro Edition: JCSPme*, Communicating Process Architectures 2006, IOS Press 2006

Codehaus Foundation (2006a), *Groovy: An agile dynamic language for the Java Platform*, Retrieved December 13, 2007 from http://groovy.codehaus.org/

Codehaus Foundation (2006b), *Closures: Informal Guide*, Retrieved December 16, 2007 from http://groovy.codehaus.org/Closures+-+Informal+Guide

Colombo, A. (2002, July 6), *How to Play Audio in an Application*, Java Media Framework, Developer Forums, Sun Developer Network, Retrieved November 2007 from http://forum.java.sun.com/thread.jspa?threadID=264317&messageID=1041697

Curnow, C.E. (2006), *Public Access to Decorative Arts Collections in Museums via Mobile Hand Held Technology – An Investigation*, Proposal and Evaluation, Master of Science in Multimedia and Interactive Systems, Napier University

Dijkstra, E.W. & Feijen, W.H.J. (1988), *A Method of Programming*, Addison-Wesley

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995), *Design Patterns: Elements of Reusable Object-Orientated Software*,

Grand, Mark (2002), *Patterns in Java: Volume 1*, Second Edition,Wiley

Hansson, D.H. (2007), *Ruby on Rails*, Retrieved December 16, 2007 from http://www.rubyonrails.org/

Hatala, M. & Wakkary, R. (2005, August 13), *Ontology-Based User Modeling in an Augmented Audio Reality System for Museums*, User Modeling and User-Adapted Interaction (2005) 15:339–380, Springer 2005, [accessed electronically]

Hoare, C.A.R. (1985), *Communicating Sequential Processes*, Prentice-Hall International

IBM, (Oct 2007), *JCSP Tutorial*, Retrieved December 13, 2007 from http://www-128.ibm.com/developerworks/java/library/j-csp1.html

Karpf, A. (2002, March 12), *Hands-on museums: just the place to switch minds off*, The Guardian, Retrieved July 24 2007 from http://proquest.umi.com/pqdweb?did=110390744&sid=4&Fmt=3&clientId=18443&RQT=309&VName=PQD

Kerridge, J. & Chalmers, K. (2006), *Ubiquitous Access to Site Specific Services by Mobile Devices: the Process View*, Communicating Process Architectures 2006, IOS Press, 2006 [accessed electronically]

König D., Glover, A., King, P., Laforge, G. & Skeet, J. (2007), *Groovy in Action*, Manning Publications Co.

Layton, J. & Franklin, C. (2007), *How Bluetooth Works*, How Stuff Works. Retrieved November 22, 2007 from http://electronics.howstuffworks.com/bluetooth.htm

Lim, A. (July 4, 2006), *Extend Your Mobile Phone's Battery Life*, CNET.co.uk, Retrieved December 11, 2007 from http://www.cnet.com.au/mobilephones/phones/0,239025953,240064190,00.htm

National Museum of Scotland, *Homepage*, Retrieved August, 2007 from http://www.nms.ac.uk/nationalmuseumhomepage.aspx

Open Symphony (2000a), Quartz. Retrieved November 21, 2007 from http://www.opensymphony.com/quartz/

Open Symphony (2000b), SiteMesh. Retrieved November 21, 2007 from http://www.opensymphony.com/sitemesh/

PDA Essentials and GPS Advisor (2007), Mobile Websites, Issue 63

Petrelli, D. & Not, E. (2004, November 8), *User-Centred Design of Flexible Hypermedia for a Mobile Guide: Reflections on the HyperAudio Experience*, User Modelling and User-Adapted Interaction (2005) 15:303–338, Springer 2005 [accessed electronically]

Red Hat Middleware (2006), *Hibernate*, Retrieved December 10, 2007 from http://www.hibernate.org/

Rocher, G.K. (2006a), *The Definitive Guide to Grails*, Apress

Rocher, G.K. (2006b, April 3), *Grails: Tag Libraries and the Power of Closures*, Retrieved December 10, 2007 from http://graemerocher.blogspot.com/2006/04/grails-tag-libraries-power-of-closures.html

Rudolph, J. (2006), *Getting Started with Grails: Rapid Web Development for the Java Platform*, InfoQ, [accessed electronically]

Sun Microsystems (2007a), *Graphics 2D*, Retrieved March 19, 2007 from http://java.sun.com/docs/books/tutorial/2d/index.html

Sun Microsystems (2007b), *Java Sound API*, Retrieved July 24, 2007 from http://java.sun.com/products/java-media/sound/

Sun Microsystems (2007c), *Java DB*, Retrieved December 12, 2007 from http://developers.sun.com/javadb/

Sun Microsystems (2007d), *Java*, Retrieved March 3, 2007 from http://java.sun.com/

Sun Microsystems (2005), *FreeTTS*, Retrieved May 15, 2007 from http://freetts.sourceforge.net/docs/index.php

Sun Microsystems (2001, October 24), *JavaSound API Programmer's Guide*, Retrieved July 24, 2007 from http://java.sun.com/j2se/1.5.0/docs/guide/sound/programmer_guide/

Tang, X. (February 9, 2007), *Grails: Using Embedded Derby Database*, Retrieved December 10, 2007 from http://xytang.blogspot.com/2007/02/grails-using-embedded-derby-database.html

Telecommunications Industry News (September 27, 2005), *Users Want Increased Battery Life in Mobile Devices*, Retrieved December 11, 2007 from http://www.teleclick.ca/2005/09/users-want-increased-battery-life-in-mobile-devices/

University of Kent (2007), JCSP Online Documentation, Retrieved December 13, 2007 from http://www.cs.kent.ac.uk/projects/ofa/jcsp/jcsp1-0-rc4/jcsp-docs/

VictorSawMa.Com, (2007, October), *Java Sound Tutorial*, Retrieved December 13, 2007 from http://victorsawma.com/index.php?op=ViewArticle&articleId=66&blogId=1

Vodafone (2007), *Mobile Internet: How Much Does it Cost?*, Retrieved December 11, 2007 from http://online.vodafone.co.uk/dispatch/Portal/appmanager/vodafone/wrp?_nfpb=true&_pageLabel=template10&pageID=MI_0005&redirectedByRedirectsImplServletFlag=true

Wah, S.C. & Mitchell, J.D. (1997, January 2), *How to Play Audio in Applications*, JavaWorld, Retrieved October 2007 from http://www.javaworld.com/javaworld/javatips/jw-javatip24.html

Welch, P. (2002a, April), *Process Orientated Design for Java: Concurrency for All* [electronic version], University of Kent, ICCS 2002 (Global and Collaborative Computing), Retrieved December 13, 2007 from http://www.cs.kent.ac.uk/projects/ofa/jcsp/jcsp.pdf

Welch, P., Aldous, J. & Foster, J. (2002b, April), CSP *Networking for Java: JCSP.net* [electronic version], University of Kent, ICCS 2002 (Global and Collaborative Computing)

Welch, P. & Brown, N. (2007), *JCSP Official Website*, Retrieved December 13, 2007 from http://www.cs.kent.ac.uk/projects/ofa/jcsp/

Whiddett, D. (1987), *Concurrent Programming: for software engineers*, John Wiley & Sons

WoTUG (2006, October 14), *The Place for Communicating Processes*, Retrieved July 24 2007 from http://www.wotug.org/

Zukowski, J. (2000, March 25), *How do I display an image in an applet or application*, JGuru, Retrieved March 27, 2007 from http://www.jguru.com/faq/view.jsp?EID=28451

Appendix 1 Project Overview

Appendix 2 Second Formal Review Output

Appendix 3 Diary Sheets & Time Boxing Schedule

Appendix 4 Communicating Sequential Processes Paper