Evaluation of the Model-View-Controller design pattern when applied to a heterogeneous application to distribute newspaper textual content to mobile devices

Sakib Supple

30 September, 2006

Department of Computing
Faculty of Mathematics, Computing and Technology
The Open University

Walton Hall, Milton Keynes, MK7 6AA
United Kingdom

http://computing.open.ac.uk
Evaluation of the Model-View-Controller design pattern when applied to a heterogeneous application to distribute newspaper textual content to mobile devices

A dissertation submitted in partial fulfilment of the requirements for the Open University’s Master of Science Degree in Software Development.

Sakib Supple
()

6 March 2007

Word Count: 13820
Preface

Thanks to my employer, Guardian News & Media Limited, for being very understanding and also for providing the substantial resources required for this project.

Thanks also to the numerous people who helped me during the project: Gavin Bresler for installing the blackberry application on numerous devices (frequently with very little notice); Nick Passmore, Lee Dunn, John Stuttle, Rob Johnson, James Bunch, Suzanne Amos and Harold Frayman, all of whom kindly agreed to help with the evaluation of the user interface.
# Table of contents

**Preface** ........................................................................................................................... 2

**List of figures** ................................................................................................................. 6

**List of tables** ................................................................................................................... 7

**Abstract** .......................................................................................................................... 8

1 **Introduction** ........................................................................................................... 9

1.1 **Problem domain** ................................................................................................... 9

1.1.1 Commercial background ........................................................................................... 9

1.1.2 Academic overview .................................................................................................... 10

1.2 **The aims and objectives of the research project** ................................................. 13

1.3 **Overview of the dissertation** ................................................................................. 14

1.4 **Contribution to knowledge** ................................................................................... 15

1.4.1 Commercial considerations ..................................................................................... 15

1.4.2 Academic contribution ............................................................................................. 16

2 **Literature review** ....................................................................................................... 17

2.1 **Bridging patterns** .................................................................................................. 17

2.2 **Design patterns** ...................................................................................................... 20

2.3 **Model-View-Controller** .......................................................................................... 21

2.4 **MVC combined with the J2EE Service to Worker design patterns** .................... 25

2.4.1 Service to Worker design pattern .......................................................................... 26

2.4.2 Modifications to Service to Worker ......................................................................... 27

2.5 **Model-Presentation-Controller-Coordinator** ....................................................... 29

2.6 **Struts framework** ................................................................................................... 31

2.7 **User interface design and evaluation** ..................................................................... 33

2.8 **Research question** .................................................................................................. 35

3 **Research methods** ..................................................................................................... 36

3.1 **Multi-channel bridging pattern** ............................................................................. 36

3.2 **User interface design** ............................................................................................ 37

3.3 **User interface evaluation** ...................................................................................... 38

4 **Data collection** .......................................................................................................... 41

4.1 **User interface evaluation** ...................................................................................... 41

4.2 **Preliminary analysis of the user interface evaluation** ........................................... 41

4.2.1 Quantitative analysis ............................................................................................. 41

4.2.2 Qualitative analysis ................................................................................................ 42

5 **Results** ..................................................................................................................... 43

5.1 **Choosing the design pattern** ................................................................................. 43

5.1.1 Constraints imposed by the Octopus Content Server ......................................... 44

5.1.2 Other constraints .................................................................................................... 48

5.1.3 Multi-channel bridging pattern ............................................................................. 48
Document Overview ......................................................................................................... 110

Functional Requirements ................................................................................................. 110
   Issue date ......................................................................................................................... 110
   Books ............................................................................................................................... 110
   Headline information ....................................................................................................... 111
   Story information ........................................................................................................... 111

Non-functional Requirements .......................................................................................... 111
   Performance .................................................................................................................... 111
   Usability .......................................................................................................................... 112

Appendix G: Multi-channel bridging pattern as specified by Folmer et al........ 113

Index .................................................................................................................................. 120
List of figures

Figure 2.1: Model with multiple Views ........................................22
Figure 2.2: Service to Worker components ...............................27
Figure 2.3: Task based model with multiple views ............28
Figure 5.1: Octopus Content Server system architecture .44
Figure 5.2: Octopus Content Server to client dataflow ......45
Figure 5.3: Mapping of tasks to views .................................54
Figure 5.4: The initial screen. StoryBrowser and MidpStoryBrowser classes ..........................................................55
Figure 5.5: The book screen. BookScreen and MidpBookScreen classes ............................................................56
Figure 5.6: The headline screen (sections). HeadlineScreen class ..............................................................................56
Figure 5.7: The headline screen. HeadlineScreen and MidpHeadlineScreen class .....................................................57
Figure 5.8: The story screen. StoryScreen and MidpStoryScreen class ..........................................................................57
Figure 5.9: Simplified class diagram for the Blackberry BookScreen view .........................................................................58
Figure 5.10: Simplified class diagram for the MIDP 1.0 MidpBookScreen view ............................................................58
Figure 5.11: Database structure for the application-device configuration ........................................................................58
Figure 5.12: Task timings for the Blackberry story browser.. ..........................................................................................68
Figure 5.13: Download rates for common tasks ............69
Figure 5.14: Data transfer rate and amount of data transferred .......................................................................................70
Figure C.1: The evaluation form .................................................90
List of tables

Table B.1: Contents of the MOBILE_DEVICES table ...... 85
Table B.2: Contents of the MOBILE_APPLICATIONS table. ................................................................................... 85
Table B.3: Contents of the APPLICATION_TASK_STATES table ................................................................................... 85
Table 5.4: Contents of the TASK_VIEW_STATES table... 85
Table C.1: The pre-session questionnaire ....................... 86
Table D.1: Performance testing for common tasks ......... 91
Table D.2: The raw evaluation data for the Blackberry application ................................................................................... 93
Table D.3: Final evaluation data ............................................. 94
Abstract

The rise in mobile device usage is fuelling demand for access to news stories while on the move. The aim of this research is to suggest a suitable design pattern for applications that publish newspaper text content to many different mobile devices. The Model-View-Controller (MVC) pattern is often used to design applications of this type. Three competing MVC-based designs are compared using the multi-channel-access bridging pattern. This tests the designs for architectural compatibility with an existing server infrastructure. The selected design pattern, a modified Service to Worker pattern, is used to produce a Blackberry application to publish news content. The user interface is designed using human-centred techniques. This interface is quantitatively evaluated and compared with the application performance requirements. It is qualitatively evaluated using a participatory heuristic inspection technique to test its usability. This evaluation identifies a number of defects in the user interface and recommendations are made to address these. To demonstrate the suitability of the design for heterogeneous applications, the code for the Blackberry version is reused to implement a generic version for mobile phones. The result of the project is a heterogeneous mobile application to publish newspaper text content.
1 Introduction

1.1 Problem domain

1.1.1 Commercial background

My employer, Guardian News & Media Limited (GNM), publishes The Guardian and The Observer newspapers, as well as the Guardian Unlimited website. The newspaper publishing industry is undergoing a revolution with many news organisations providing editorial content on the Internet. Traditionally this content has been delivered to desktop personal computers connected to the Internet using wired connections (WAN, LAN or broadband connections). With the recent rise in mobile device usage and forecasts (GSM World, 2006) suggesting that this trend will continue, mobile access to news content is an important area within GNM’s strategy. A vital aspect of this is that users should be able to read news on various different mobile devices: mobile phones, PDA’s and Blackberries. In particular, the Blackberry device has increased in popularity both as a personal communication device and as a corporate tool within GNM, with senior editors, senior managers and executives using them for email, intranet and internet access. There is a requirement for an application (called the Story Browser) that will provide editors with mobile access to the articles published by GNM, and that could in the future be offered to readers as a subscription service capable of being deployed on a number of different mobile devices.

This leads to the requirement for an application that will, in the first instance, be deployed on Blackberry mobile devices, but will exhibit enough heterogeneity to also run on other mobile and desktop devices.
1.1.2 Academic overview

The field of ubiquitous and pervasive computing (Ubicomp) includes a significant amount of research related to device-independent, heterogeneous software development (Niemelä and Latvakoski, 2004). The requirements of Ubicomp are greater than just client-side device-independence including interoperability, heterogeneity, mobility, adaptability, service discovery and context-awareness. Some of the research dealing with these problems involves middleware like Capanet (Davidyuk et al., 2004) with other research investigating approaches like Model Driven Architectures (MDA) (Alexandre, 2003; Rouvoy and Merle, 2004).

In the case of the GNM story browser application, the server architecture is already in place. The workflow and text data for all GNM publications is stored in two Oracle databases. As newspaper and magazine pages are sent to the press sites to be printed, their text is extracted and stored in one of the databases (the text archive), while the other database (the workflow database) contains information about when that content was published as well as information about the page number and publication and section that it belongs to. A server process has been developed, called the Octopus Content Server, that on receipt of a request from a client application, pulls together publication, section and story data from the two databases in a coherent form and sends it back to the requesting client. The client application will have to communicate with the server using TCP sockets. The server makes no assumptions about the nature of the client application or the device it runs on, however it does return the data to the client in a proprietary format in a GZIP compressed stream.
The research to be carried out revolves around the client application. That application should operate on multiple devices and must be developed within the constraints of the existing server interfaces.

Although the user interface of each client device is different, that is to say the view of the data that a user will see is different on each device, the actual text data itself is the same for all devices. A popular design pattern for this type of multi-channel application, where many different views are required of a fixed data set, is the Model-View-Controller (MVC) design pattern (Folmer et al., 2006; Gamma et al., 1997, pp.4-6; M878, 2000, pp. 13-15). A number of extensions to the MVC design pattern have been suggested for this type of mobile application. For example:

- MVC combined with the J2EE Service to Worker design pattern (Sauter et al., 2005)
- Model-Presentation-Controller-Coordinator (MPCC) (Cerqueira et al., 2001; Hess and Campbell, 2003; Hess et al., 2002)
- Struts framework (Jing-Mei Li et al., 2006)

The research involves the survey and analysis of these existing MVC-based design patterns, and a discussion of how one of them (or if none of them is appropriate, a suitably modified version of one of them) could be applied to GNM’s story viewing application. Developing and deploying a prototype application on the Blackberry platform will test the design, and the compatibility of the design to other platforms will be demonstrated. The
usability of the Blackberry interface will be evaluated by distributing the prototype to a small group of users and gathering suitable metrics.

The evaluation of the design will indicate the suitability, or otherwise, of the MVC-based design pattern for heterogeneous applications.

The evaluation will involve the definition of a set of functional requirements and non-functional requirements for the application. The design patterns will be evaluated using a technique proposed by Folmer et al. (2006) for bridging patterns and the results used to compare them to determine the most appropriate one. Folmer et al. suggest that bridging patterns provide a standardised mechanism for user interface designers and software engineers to communicate. One feature of the technique is that it allows user interface designers to judge if their proposed interface is compatible with the anticipated, or existing, software architecture. This feature is particularly useful for this project. This technique will provide a mechanism to test a user interface design against the constraints of the existing software architecture without the need for coding any applications. This differs from Folmer et al.’s use of the technique: they use it as a communication mechanism between user interface designers and software developers to isolate, early in the development process, any problems for the software architecture introduced by interface design decisions. For this project it will provide a set of standardised criteria that can be used to compare competing MVC design patterns. In the context of this project the technique allows possible designs based on the three competing design patterns to be compared and evaluated without having to actually code any applications; there is insufficient time and
resource to create an application for each design. A disadvantage of the approach is that the results may not be as accurate compared to actually coding applications using the different design patterns, then evaluating each of them separately.

Once a design pattern has been chosen, it may be necessary to modify the design in some way. This final design will then be used to create an application that can be evaluated by users (M873, 2001, Units 3, 4 and 5) to see how well the usability and functional requirements of the application have been met.

1.2 The aims and objectives of the research project

Mobile devices with access to the Internet are becoming ever more popular, and this popularity is set to increase. These mobile devices vary greatly: different CPUs, screen sizes, screen resolutions, keyboards, input mechanisms and connectivity capabilities.

Internet access on these devices is often used to read news content. The problem this raises is how to support the myriad of different devices available. The ideal scenario for a content supplier is to be able to develop a single application that will operate on different devices, while still making the most of the strengths of each device and providing the user with a good quality of service and a usable interface. In an ideal world the application would function on all current and future mobile devices without any code changes, but this will not always be possible; however the application should be designed to keep changes of this type to a minimum.
Investigating the utility of the Model-View-Controller (MVC) design pattern in producing such an application is the aim of the project. A number of different MVC-based design patterns have been suggested as patterns capable of producing good software designs for this type of system. Three of these patterns will be analysed and investigated to ascertain their usefulness for this purpose. This will build on the previous research done on them.

The aim of the project is to determine a suitable MVC-based design pattern for the design of heterogeneous mobile applications.

1.3 Overview of the dissertation

The dissertation is structured in the following way:

- A literature survey is carried out to identify suitable candidate MVC-based design patterns for the design of a heterogeneous mobile application. The survey proposes user interface evaluation techniques to ascertain the usability of the application.

- The bridging pattern technique is used to compare the candidate design patterns to identify the most appropriate one for the project application.

- The application is designed and coded for the Blackberry platform and the resulting user interface evaluated using participatory heuristic evaluation (Muller et al., 1998) techniques.

- The results are analysed and a conclusion reached as to the success of the user interface.
• The application is deployed on another platform and a conclusion reached as to the heterogeneity of the software

1.4 Contribution to knowledge

1.4.1 Commercial considerations

Mobile Internet access to news content is considered by many media organisations to be an important aspect of their strategy.

It has been identified that internal users at Guardian News & Media Limited (GNM) (editors and correspondents deployed all over the world) need mobile access, via their Blackberry devices, to stories published by GNM in a format that is familiar to them; that is, structured as they appear in a newspaper. The application developed during this project will provide this and allow them to monitor the production of all newspaper and magazine products. This allows increased collaboration and productivity between editors and correspondents. From a commercial point of view, this application could be used to offer text content from GNM products as a subscription service to mobile users. This would open up new revenue streams for selling advertising as part of the mobile service. Any subscription and advertising revenues generated can be maximised by reaching as many mobile users as possible. This leads to the requirement for the application to support the numerous mobile devices currently in use as well as those that will be developed in the future.

This research will therefore be of interest to any organisation that plans to make text content available to staff and customers on a mobile platform.
1.4.2 Academic contribution

This commercial requirement can be met by producing a heterogeneous application that can run on many devices with minimal changes. This research shows whether this is possible using MVC-based design patterns, and proposes an appropriate pattern for this purpose.

This area is the subject of considerable academic research for ubiquitous and pervasive computing, and heterogeneity of applications plays an important role in producing ubiquitous applications (Niemelä and Latvakoski, 2004). The project builds on research carried out on the use of the MVC design pattern to provide heterogeneity in ubiquitous and pervasive computing, by analysing how this design pattern can be applied to the specific application of the dissemination of newspaper text content to mobile users. The project provides a critique of some of the MVC-based design patterns in current use, and further evidence of how MVC can be applied in this area. It may be of interest to researchers and software designers in the area of ubiquitous and pervasive computing.

The user interface was designed using human-centred techniques based on the ISO 13407 (Earthy et al., 2001) international standard. The interface was then evaluated using participatory heuristic evaluation techniques (M873, 2001, Unit 6, pp. 43-60; Muller et al., 1998). It may therefore be of interest to those researching topics related to Human-Computer Interaction and the evaluation of user interfaces.
2 Literature review

This section will detail the literature search carried out for this project and present a review of the current body of knowledge relevant to this research.

Research related to bridging patterns (Folmer et al., 2006; Trætteberg and van Welie, 2000; van der Veer and van Welie, 2000) is outlined, with emphasis to their use in assessing the applicability of user interface design patterns, taking into account architectural constraints, without having to actually write any application software to make that assessment. In particular the multi-channel bridging pattern is reviewed as a mechanism to evaluate the three candidate MVC-based design patterns assessed for the project application.

There is a definition of what is meant by a design pattern, explaining how design patterns can be useful for projects of this type. The general Model-View-Controller (MVC) design pattern is reviewed, followed by a discussion of why it is relevant to this research. Three variants of MVC that have been used to solve problems similar to those faced by this project are introduced and reviewed in detail.

Finally there is a discussion of user-centred user interface design, and the evaluation of user interfaces using heuristic evaluation techniques.

2.1 Bridging patterns

A general problem has been identified in the design and implementation of software (Folmer et al., 2006; Trætteberg and van Welie, 2000): user interface designers often do not take into account the architectural constraints that software engineers will face when implementing a design, and this leads to expensive re-
working late in the project life-cycle. Folmer et al. (2006) suggest an approach to solving this problem using ‘Bridging Patterns’ that bridge the gap between the UI designer and the software engineer, by allowing the UI designer to carry out an architectural analysis of the design before implementation begins.

In the same way that Gamma et al. (1997, p. 6) use a standard template to describe design patterns, Folmer et al. (2006) suggest a standard format to be used to describe a bridging pattern. This consists of seven sections:

- **Problem**: problems related to system usage that are relevant to any stakeholder that is interested in the usability of the system

- **Use when**: A situation that gives rise to a usability problem in terms of tasks, users and context of use

- **Solution**: A proven solution to the core of the problem

- **Why**: How and why the pattern works including an analysis of how it affects various attributes of usability

- **Examples**: An example of how the pattern has been applied to a real system

- **Architectural implications**: An analysis of the architectural impact of the pattern and of the responsibilities that must be fulfilled by the architecture. This is particularly important for this project since the mobile application must operate within the architectural constraints of the Octopus Content Server
• **Implementation:** Implementation details in terms of the classes, technologies or techniques that must be used.

The key features of this approach are that the bridging pattern provides (Folmer et al., 2006):

- Detailed implementation issues and solutions for a design pattern allowing the designer to assess the architectural impact of implementing that pattern prior to any development work. This is especially important for this project

- Better communication between user interface designers and software engineers

Folmer et al. present a multi-channel access pattern. In the ‘problem’ section of their definition of this bridging pattern they state that the problem that it can solve is:

'A user wants or requires access to a system using different devices (mobile phone, desktop, PDA)'

This project involves finding a suitable design pattern to solve this exact problem. The relevant sections of the bridging pattern are Architectural Implications and Implementation. The bridging pattern method is used to analyse each of the three competing MVC design patterns in terms of the architectural implications and implementation details of using that pattern. This provides a standardised set of criteria that are used to compare the competing MVC design patterns for compatibility with the software architecture at GNM, without having to write any code. This differs from the way Folmer et al. use the technique. Here it is used to
compare designs, whereas Folmer at al. use it as a mechanism for communication between user interface designers and software engineers. Where the two approaches are similar is that in both cases the bridging pattern helps to identify design conflicts at an early stage in the development process.

Folmer et al. suggest that an MVC design pattern could provide a solution to the problem of multi-channel access, and they analyse some of the architectural and implementation details when using the Struts framework for this purpose.

The same approach is applied here to the modified Service to Worker and the MPCC design patterns. This provides standardised points for comparison between the design patterns. This forms an important part of my work since the interface produced must function within the constraints of the server architecture at GNM.

**2.2 Design patterns**

The first issue to consider is why and how design patterns are used. Software designers working in different domains often encounter design problems that are similar. For example, an accounting package may contain two objects that need to communicate, but have incompatible interfaces; the same problem may exist in a flight simulator application. Although the application domains are different, the software design problem and its solution are the same: adapt the interface of one object so that it is compatible with the other object.

This suggests the need to capture the experience of software designers so that general solutions to recurring problems can be harnessed and reused by others. Design patterns achieve this by expressing successful designs and architectures
in a uniform way. These patterns allow designers of varying experience to evaluate different solutions to design problems for a particular application.

This project uses the definition of a design pattern suggested by Gamma et al (1997, p. 3):

‘Design patterns…are descriptions of communicating objects and classes that are customised to solve a general design problem in a particular context’

Gamma et al also suggest a consistent format for describing design patterns using a template that makes the patterns easier to learn, compare and use (Gamma et al., 1997, p. 6). This template requires information about the pattern related to issues like motivation, applicability, structure, participants, collaborations, consequences and implementation.

In the case of this project the generalised problem is how to deliver and display the same information in different formats in a heterogeneous environment. The Model-View-Controller (MVC) design pattern is often used to solve this general problem, and this is investigated further below.

2.3 Model-View-Controller

The Smalltalk-80 programming language makes use of three cooperating classes to build and manage user interfaces: Model, View and Controller (Krasner and Pope, 1988). Each object has specific responsibilities (Gamma et al., 1997, pp. 4-5):

- **Model**: contains the application data
- **View**: the screen representation of the model
• **Controller:** defines how the user interface reacts to user input

This class structure decouples the three components making them more flexible and easier to reuse. In particular a single Model can be represented by multiple Views. In this project a single story could be displayed on different platforms. See Figure 2.1 – this diagram is similar to one used by Gamma et al (1997, pp. 4-5).

![Diagram showing single Model with multiple Views](image)

Figure 2.1: Model with multiple Views

Each platform is represented by a different View. Using an example Model that describes a newspaper story, the essential features of this approach are that:

- Any changes to the story are reflected in each view; it is the responsibility of each view to ensure that its current appearance reflects the current state of the story (the Guardian produces five editions of the printed newspaper a day, so a story may change over time)

- Each separate view of the story is independent of other views
• A subscribe/notify protocol is used: each story view subscribes to notifications of change; when the model changes, all views are notified of that change

• Nested views are supported. This would allow the publication-book-section-page-headline structure to be represented on screens of limited size (see page 24 for an explanation of the newspaper structure)

The subscribe/notify relationship between the Model and View is described in a more generalised way by the Observer design pattern (Gamma et al., 1997, pp. 293-303). In the context of this project the ability to notify views is not relevant: the requirements of the application specify that data will be ‘pulled’ from the server on user-request, and there is no requirement for that data to be updated after this initial interaction. The MVC architecture does allow this functionality to be added later if required. However, the concept of a single model feeding multiple views does fit well with the concept of heterogeneity by allowing the same story data to be viewed on different devices that have quite different hardware profiles (screen size and resolution, keyboard input, application memory and so on).

The ability to create nested views is described more generally by the Composite design pattern (Gamma et al., 1997, pp. 163-173). This pattern provides a more generalised way to handle nested objects. This aspect of the architecture was less relevant to this project, although it is a useful feature of the pattern when considering how to break down complex views when displaying them on screens of different sizes – for example to represent the hierarchical structure of a newspaper made up of many books (the main newspaper, a features section like
the Guardian G2 or Media section and so on), each made up of one or more sections (Home News, Foreign News, City & Finance and so on), each section containing one or more pages with each page having one or more headlines.

The Controller encapsulates how a View may respond to user input, and the essential features of the View-Controller relationship are (Gamma et al., 1997, pp. 4-5):

- It is possible to change the way a view responds to user input without changing the visual representation of that view
- It is easy to create variations of existing controllers
- The controller for a view can be changed at runtime

As suggested by Cerqueria et al. (2001), an interesting extension of the Controller would be to allow it to control the View’s response to input from sources other than the user: for example in response to some change in the operating environment of the application, or to a notification of change to the model. This is considered an important requirement for Ubiquitous computing (Cerqueira et al., 2001; Hess and Campbell, 2003; Hess et al., 2002).

The View-Controller relationship can be described in a more generalised way by the Strategy design pattern (Gamma et al., 1997, pp. 315-323). It defines an object that can contain an algorithm, and that algorithm can be changed dynamically. This provides a mechanism for dynamically changing the way that a view is displayed so that best use can be made of device-specific UI components. This is useful for this project.
It can be seen therefore that the core MVC pattern is a combination of the Observer, Composite and Strategy design patterns. Various aspects of these patterns are relevant, to differing degrees, to the design of the software developed for this project.

Because of the features of MVC described above, a number of extensions to the pattern have been proposed specifically for use in heterogeneous, mobile environments. Three of these are now reviewed in the context of the application to be developed for this project.

2.4 MVC combined with the J2EE Service to Worker design patterns

One specific issue that needs to be resolved when developing user interfaces for multiple devices is the following (Sauter et al., 2005): how a dialog containing a complex form (many menus, input fields and so on) that can be displayed on a single page when running an application on a desktop computer, can then be displayed in several separate dialogs on a device with a smaller screen such as a Blackberry or mobile telephone.

Sauter et al. (2005) have suggested combining MVC with a modified version of the Java 2 Enterprise Edition (J2EE) Service to Worker design pattern to add device independence to Java web applications. This is achieved by adapting the interface dynamically to meet the requirements of the specific device displaying the content. In this way a large complex form can be split into smaller forms spread over multiple pages that are more appropriate for a particular device.
2.4.1 Service to Worker design pattern

First consider the basic Server to Worker design pattern. This design pattern is a combination of two other design patterns in the Sun Microsystems online pattern catalogue (Service to Worker, 2002): Front Controller (Front Controller, 2002) and View Helper (View Helper, 2002).

The controller receives the initial request for a view. It manages content retrieval and may require helpers to perform all required functions. The controller delegates the job of managing views and navigation to a dispatcher. Again the dispatcher may need helpers to carry out all the required functionality. These helpers dynamically generate the requested view. Although the underlying model may remain unchanged, a helper component can generate an intermediate model that is adapted for a specific view. The helpers can either provide the raw data to the view, or they may reformat the data for display, for example as HTML. The controller may encapsulate a dispatcher or there may be a separate dispatcher component. A generalized diagram is shown in Figure 2.2.

In this way, it can be seen that the Server to Worker pattern is based on MVC and attempts to encapsulate a similar relationship between application data and its screen representation.

The Sun Microsystems online pattern catalogue (Service to Worker, 2002) states that the pattern forces:

- Business logic to be encapsulated in components other than the view
- Multiple views potentially map to the same request
This is equivalent to the model containing the application state and data, with multiple views representing that data: a stated feature of MVC. This fits in with the requirements of this project.

### 2.4.2 Modifications to Service to Worker

Sauter et al. (2005) modify this design pattern by introducing a task-based approach to the design method. This is based on a suggested model from Banavar et al. (2000) whereby the task logic, or task flow (comprising of a controller, dispatcher and one or more helpers), is separated from one or more views. This task flow can be thought of as the single model represented by multiple views. A generalised diagram is shown in Figure 2.3 below.
This approach requires that the application be designed using a task-driven methodology. Each task is represented by a single view. If a device cannot support the single view for a task (for example the device screen is too small to display all of the information on a form), then more views must be assigned for that task, and the most appropriate view is selected at runtime. The modified Service to Worker model achieves this by introducing two additional components: a TaskStateBean that works with the dispatcher and a ViewStateBean that works with a view. A single view component (for web services this means a single Java Server Page) must be created for every combination of task and device to be supported. The TaskStateBean helps to manage the navigation by determining which view to display next in the context of the current task state. The ViewStateBean contains information about which elements to display for a given task and device, so this is queried before the final view is rendered.
It can be seen therefore that this approach could be used to create a heterogeneous mobile application capable of being deployed on multiple platforms with minimal code changes. It may therefore be useful for this project.

2.5 Model-Presentation-Controller-Coordinator

Another important issue in pervasive and ubiquitous computing is how an application will function in situations where its context changes (Hess et al., 2002): for example the device it runs on and the environment in which it operates (affecting aspects such as screen size, memory usage, network connectivity) may be subject to frequent change. In particular there has been an increase in the use of mobile devices, so the ability of an application to operate in a device-independent way is important (Niemelä and Latvakoski, 2004).

Hess et al. (2002) introduce the concept of a device-independent User Virtual Space as a mechanism to meet these needs. It moves with the user and consists of data, tasks and devices associated with that user. This virtual space must be context-sensitive and map data and tasks into the user's environment as the context changes.

To support this, Hess et al propose a middleware solution that contains the resources required together with a set of standard interfaces and mechanisms to co-ordinate those resources. They call the environment supported by this middleware an 'active space': an entity that has a specific set of resources available to it (for example a plasma screen attached to a desktop PC connected to the internet via a LAN, or a mobile telephone with a low resolution screen a few square centimetres in area connected to the internet via GPRS). For the purposes
of this project, the notion of an active space could refer to different mobile devices: entities with different resources available to them (screen size, memory profile, network connectivity, input mechanisms) such as a Blackberry, a mobile telephone or a desktop computer with a large monitor.

To create a prototype active space, and the middleware to support it, a variant of the MVC design pattern has been proposed: Model-Presentation-Controller-Cordinator (MPCC) (Hess et al., 2002). This differs from the standard MVC approach in that the Presentation is a generalisation of the View, and the Coordinator additionally co-ordinates the activities of the presentation and controller components.

The breakdown of the roles and responsibilities of the four components is:

- **Model**: contains the application data, state and logic (as in MVC)
- **Presentation**: a generalisation of a view that exports the state of the Model
- **Controller**: defines how the user interface reacts to input events in the same way as in MVC. The input event could be a context change (user moves from one ‘active space’ to another) as well as more traditional user-generated events (key press, mouse click, sound command)
- **Coordinator**: this coordinates the other three components. It can connect or disconnect a controller or presentation and maintains a list of the current presentations.
The application is described using two different types of file: Application Generic Description (AGD) and Application Customized Description (ACD). The AGD is device-independent and contains information required to make the application run regardless of the context. The ACD customizes an AGD for a specific device, customizing the application instantiation process for the available resources: only the components required for the specific device are instantiated. This allows multiple devices to be supported by creating separate ACD’s for each device using the same AGD.

Although the problem that Hess et al are trying to solve is more complex than that faced by this project, a fundamental part of their work involves building heterogeneous applications. This solution requires a piece of middleware that is treated as an operating system. Despite this, the MPCC design could still be relevant to the design of a client-side user interface for a device-independent application for this project. This is because the design pattern has the essential features required:

- The Model contains the application data and is decoupled from its screen representation
- Multiple Presentations can be used to represent a single model

### 2.6 Struts framework

The next candidate design pattern is the Struts framework. This is an open source framework that combines Java web development with MVC (Apache struts, 2006; Struts user guide, 2006). It is maintained by the Apache Struts Project (Apache
struts, 2006) and was launched in May 2000 to provide a standard MVC framework for Java developers (Struts user guide, 2006).

Prior to the introduction of Struts, Java web developments involved writing Java Server Pages (JSP): a mixture of HTML and Java code, the Java code being written in-line. Similarly, Servlets (modules of Java code that run in an application server to answer requests from client applications) could be written that were difficult to maintain and reuse since they could contain HTML rendering code in them. This led to business logic, flow control and the presentation of data all being mixed up in a single entity: the Java Server Page or Servlet code (Apache struts, 2006; Jing-Mei Li et al., 2006; Struts user guide, 2006).

The need to make code more flexible and reusable means that the presentation of data has to be de-coupled from the business logic and flow control. The Struts framework achieves this by separating the responsibilities for each as follows:

- **Model**: an appropriate technique to access the application data can be used for each application (Struts user guide, 2006), and could include Servlets using JDBC or Java Beans. This project could use Java Beans to connect to the Octopus Content Server to return story data.

- **View**: The Struts Taglib component takes care of data presentation allowing model and view to be de-coupled. The relationship between Model and View is managed through user-defined tags (Jing-Mei Li et al., 2006). The actual presentation of data can be carried out using various technologies: Java Server Pages, Cocoon, Extensible Style Sheet Language (XSLT) and so on.
• **Controller**: this is the ActionServlet. It reads a configuration file mapping requests to the Action class (Model) that should honour that request (Jing-Mei Li et al., 2006; Struts user guide, 2006). This is also responsible for mapping the classes for form validation.

This design fits in with the need for a single model feeding multiple views, and so could have been relevant to the design of a heterogeneous mobile application.

### 2.7 User interface design and evaluation

The usability of a computer system is considered to be critical its success. A widely adopted definition of ‘usability’ is that from Nielson (Nielsen, 1993; M873, 2001, Unit 6, pp. 43-60): the usability of a system is concerned with how learnable, efficient, error-free, memorable and satisfying to use the user interface is. It has been recognised for some time that due to the limitations of mobile devices (limited screen size and resolution, connectivity, lack of mouse input) that effective user interface design is especially important for mobile applications (Buchanan et al., 2001).

Human-centred techniques for user interface design have proved successful as an approach to designing usable interfaces (M873, 2001, Unit 6, pp. 43-60), and international standards have been introduced, such as ISO 13407 (Abran et al., 2003; Earthy et al., 2001), to encourage their use. The advantages of this approach can be summarised as (M873, 2001, Unit 3, pp. 32-33): reduced training and support costs due to the system being easier to use and understand; improved user satisfaction; efficiency and productivity improved; and possible improved competitive advantage due to better quality and aesthetics in the
product. All of these are relevant to this project. Furthermore, there has been some success in adapting these human-centred techniques to deal with the constraints of the mobile environment (Buchanan et al., 2001).

Once a user interface has been created, improving it involves evaluating its usability as part of an iterative design process (M873, 2001, Unit 6, pp. 43-60). One approach is by usability inspection, and in particular Nielsen’s approach of Heuristic Evaluation (Nielsen, 1993; M873, 2001, Unit 6, pp. 43-60) has been widely used. This approach involves finding problems in the design without carrying out costly usability tests that require a large set of resources, known as a discount method, (Nielsen, 1993; M873, Unit 6, pp. 43-60). The technique achieves this by having a small number of domain experts examine the interface and judge its compliance with a set of recognised usability principles known as heuristics. This mechanism is effective but some problems have been identified with it (M873, 2001, Unit 6, pp. 43-60): the domain experts may not predict what a real user might do when judging compliance of the interface to the heuristics; domain experts may attach importance to something that is irrelevant to real users thereby misleading the development process. These problems are addressed by a variant of Nielsen’s approach proposed by Muller et al. (1998): Participatory Heuristic Evaluation. For this project, the essential differences between this and Nielsen’s approach are:

- There is a modified list of ten heuristics
- The judgement of how well the interface follows the heuristics is made by real users of the system rather than domain experts
Using this technique provides an effective mechanism for evaluating and improving an application interface.

### 2.8 Research question

The aim of the project is to propose an MVC-based design pattern for the design of heterogeneous mobile applications that publish newspaper text content.

Using the research methods, design patterns and evaluation techniques outlined above, it should be possible to design, implement and evaluate a heterogeneous mobile application for viewing newspaper text content on more than one type of mobile device.

Furthermore it should be possible to reach a conclusion as to the suitability of the chosen design pattern in producing such an application.
3 Research methods

The research question defined above required both an analytical and an empirical approach, and therefore required both qualitative and quantitative analysis.

As a starting point, a requirements specification for the application was produced. The specification is shown in Appendix F.

3.1 Multi-channel bridging pattern

The literature survey was used to isolate three MVC-based design patterns used to provide heterogeneity for mobile applications (Hess et al., 2002; Sauter et al., 2005; Struts user guide, 2006). The information collected was used to place the project in context. These design patterns were analysed using a similar technique to that proposed by Folmer et al. (2006), for bridging patterns, and evidence presented to suggest which of these design patterns could be used for the GNM Blackberry application. The constraints of the existing systems and technical infrastructure at GNM had a bearing on this. The bridging pattern method has the following relevant features:

- A standardised template is used to describe each design pattern allowing a meaningful comparison to be made without any coding

- Clearer communication of how an interface design will sit within an existing software architecture

Due to a lack of time it was not possible to code and compare applications for each design pattern. However, the multi-channel bridging pattern provided a mechanism to make a meaningful comparison of the different designs, while at
the same time testing the design for architectural with the existing infrastructure within which the application must function.

### 3.2 User interface design

The interface was designed using human-centred techniques for interactive systems based on the international standard ISO 13407 (Earthy et al., 2001). The relevant benefits of using this approach are defined in M873 (2001, Unit 3, pp. 32-33) as:

- Reduced training and support costs due to systems being easier to understand and use
- Improved user satisfaction leading to a reduction in discomfort and stress
- Operational efficiency and user productivity improved
- Improved product quality and aesthetics perhaps leading to competitive advantage

These benefits are relevant to this project for internal users of the application, and also for customers of GNM if the application were to become a commercial subscription product. The standard suggests the following essential elements of human-centred design (M873, 2001, Unit 3, pp. 32-33; Usability Net, 2006):

- Understand and specify the context of use and the early involvement of users – a clear understanding of the users, their tasks and requirements
- Specify the user and organisational requirements – isolate the functions that are carried out by users from those carried out by the system
• Produce design solutions – an iteration of design solutions driven by user feedback

• Evaluate designs against requirements – evaluate the effectiveness of the final design

These elements are especially relevant to this project since it fits well with the general approach to interface development at GNM.

More specifically, the following usability factors were taken into account during the design process (Trætteberg and van Welie, 2000):

• Visibility

• Choice of an appropriate metaphor

• Effective use of text formatting and colour

• Feedback

### 3.3 User interface evaluation

The application user interface was evaluated using a variant of the participatory heuristic evaluation techniques proposed by Muller et al. (1998) and outlined in M873 (2001, Unit 6, pp. 43-60). This involved defining the usability metrics to be measured based on the functional and non-functional requirements of the application; defining an evaluation plan that includes the elements of the evaluation, the participants in the evaluation, the tasks to be carried out, where and how the evaluation will be conducted. Only high-fidelity prototypes were used for the evaluation. This can be considered to be a validation or summative
evaluation as defined in M873 (2001, Unit 6, p. 14), since it tested the whole system including user documentation, usability and performance.

The main elements of the evaluation process included:

- Performance usability metrics were defined based on the non-functional requirements of the system. Quantitative tests were carried out to measure performance: various operations were timed and a comparison made between the results and the requirements specification.

- A pre-test questionnaire was given to evaluation users before a prototype was delivered to them for the first time. This helped to place their experience and familiarity with mobile devices into context. This was important since it had a bearing on their perception of the usability of the interface.

- A task list was prepared and the participating users asked to perform them. A questionnaire was prepared based on the evaluation heuristics together with the specific nature of the interface. Users were asked to fill it in as they carried out specific tasks.

- The effectiveness of the interface was measured by counting the number of times the heuristics were violated.

The advantages of using real users for the evaluation rather than expert users are (M873, 2001, Unit 6, p. 47):

- Expert users might not predict accurately what a real user might do
• Expert users may attach importance to something that is irrelevant to real users, therefore misleading the development process

The user interface was evaluated by a group of editorial users. This sort of approach can suffer from the subjectivity of individual users, so care was taken to use evaluation techniques that mitigate this problem.

There are over 200 Blackberry users at GNM so it was possible to recruit participants for this evaluation. The dissemination of text to mobile devices is currently a high priority topic for senior management within the organisation. This means that these participants were allowed to take time from their normal duties to take part in the evaluation.

Conclusions from the evaluation were presented and used to judge whether or not the selected design pattern was effective for this type of heterogeneous application.
4 Data collection

This section contains information about the data sources used for, and the preliminary analysis of, the user interface evaluation.

4.1 User interface evaluation

Performance tests were carried out to see if the application met the non-functional usability requirements stated in the requirements specification:

- Book list retrieved within 4 seconds
- Headline information retrieved within 8 seconds
- Story information retrieved within 8 seconds

These tests were carried out over several days in a number of locations.

The participatory heuristic evaluation was carried out according to an evaluation plan as outlined in section 5.2.

4.2 Preliminary analysis of the user interface evaluation

4.2.1 Quantitative analysis

Three common tasks were timed to measure the performance of the application: get the book information for an issue date; get the headlines for a book; get the story for a headline.

The tasks were performed ten times and a mean time calculated. The tasks were timed in three different locations in an attempt to equalize the affect of different levels of mobile service experienced. The same issue date, book and story were used for the timings in all cases. The number of bytes transferred was noted for
each task so that a download rate could be ascertained. In all cases the tests were carried out using the same Blackberry device.

The results of the performance tests are summarised in D.1 in Appendix D.

4.2.2 Qualitative analysis

A participatory heuristic evaluation was conducted by nine users. Each user was provided with a copy of each of the forms shown in appendix C:

- Pre-session questionnaire
- Definition of heuristics
- Three evaluation forms (one form per task)

In addition users were also supplied with the user documentation for the application.

Each user returned the forms as he or she completed the evaluation and the data summarised in Table D.2 in Appendix D.

The table includes comments from the evaluation form as well as the information from the pre-session questionnaire from the user who made each comment. This helps to put the comments into context.
5 Results

This section contains the results of the research.

It states the design pattern chosen to design the application, showing how the decision to use that design was reached. It goes on to discuss how the design was implemented for the Blackberry mobile device. A MIDP 1.0 version was also implemented to demonstrate the heterogeneity of the application. Finally it discusses the user interface evaluation plan, and then presents the results of that evaluation for the Blackberry version of the application.

5.1 Choosing the design pattern

There were three competing design patterns:

- MVC combined with the J2EE Service to Worker design pattern (Sauter et al., 2005)

- Model-Presentation-Controller-Coordinator (MPCC) (Cerqueira et al., 2001; Hess and Campbell, 2003; Hess et al., 2002)

- Struts framework (Jing-Mei Li et al., 2006)

These were evaluated for their ability to design a heterogeneous application capable of operating within the existing server architecture at GNM.

First, the constraints imposed by the existing software architecture, in particular Octopus Content Server were defined.
5.1.1 Constraints imposed by the Octopus Content Server

The Octopus Content Server is a multi-threaded server process, written in Java and running under the Solaris operating system. It provides client applications with story text as it was printed in the Guardian and Observer newspapers.

The system architecture is shown in Figure 5.1.

![Figure 5.1: Octopus Content Server system architecture](image)

The story data is stored in two oracle databases, one containing workflow information about which stories were actually printed (the Octopus workflow database), and the other containing the text content of the stories (the text archive database). There is an oracle database link between the two databases so that the server only has to connect to the Octopus workflow database. All the PL/SQL code to access text in the text archive database runs as stored procedures and functions in the Octopus workflow database.
The client application for this project, called the Story Browser, operates on a Blackberry handheld, and communicates with the server using TCP sockets. All matters relating to security, for example ensuring that only applications and users with sufficient privileges are allowed access to the system, are handled by the Blackberry Enterprise Server (BES) and will not be discussed in detail here. However a brief explanation of the part played by the BES in controlling the flow of data is provided.

If a wireless network is available, the Blackberry sends a request for data to the BES via the Internet. The Mobile Data Services (MDS) module of the BES then forwards the request to the Octopus Content Server on the GNM LAN. There is also a standby Content Server to provide redundancy. The TCP connection between the client application is managed by the BES, and the nature of the underlying wireless network is hidden from the both the server and the Story Browser application. The client applications ‘pulls’ data from the server in a stateless way: that is, the server does not hold any state information about any client that requests information form it, and the server never notifies the client of any changes made to the underlying data.

The data flow for the communication between the client and the server is shown in Figure 5.2:
The protocol to make a request used a simple ASCII string to define the request. The first part of the string is a single word that identifies the type of request, for example ‘getpublishedbooklist’ to get a list of all the books (physical newspaper sections) published on a specified date. This is optionally followed by the hash character ‘#’ and a list of parameters for the request, each separated by the hash character. This type of protocol was used since it is very succinct and therefore appropriate for a low bandwidth mobile network. Ideally the data is returned back to the client as a string that can be compressed using the GZIP algorithm. Since a single story can be as long as 5000 words (about 30000 bytes), compressing the data provides a significant increase in performance in low bandwidth environments. However, since not all devices can support GZIP compression, the Octopus Content Server protocol supports two sets of calls: one set that returns an uncompressed stream, and one that returns compressed data.

The recognised server commands are:

- **getpublishedbooklist#YYYYMMDD** where YYYYMMDD is the issue date. This returns a list of names and book ID’s (uniquely identifying a particular book on a particular issue date).

- **getpagesandheadlinescomp#BOOKID** where BOOKID is the unique identifier returned by the above call. This returns a list of pages, the sections they belong to and the headlines on each page. The headline data contains the text of the headline, and a unique identifier for the story that the headline belongs to. The data is passed to the client as a GZIP compressed stream.
• getpagesandheadlines#BOOKID where BOOKID is the unique identifier returned by the getpublishedbooklist call. This returns a list of pages, the sections they belong to and the headlines on each page. The headline data contains the text of the headline, and a unique identifier for the story that the headline belongs to. The data is passed to the client as an uncompressed stream.

• getstorycomp#STORYID where STORYID is the unique identifier for the story as returned by the above call. This returns the headline, byline and main body text for the requested story as a GZIP compressed stream.

• getstory#STORYID where STORYID is the unique identifier for the story as returned by the getpagesandheadlines call. This returns the headline, byline and main body text for the requested story as an uncompressed stream.

• gettaskviewsforappanddevice#APPLICATION#DEVICE where APPLICATION is the name of an application and DEVICE is the name of a device. This returns the list of class names of the views for each task in the order that the tasks are executed. This call was added to the server specifically for this project. The data is returned as an uncompressed stream.

To summarise, the constraints imposed by the Octopus Content Server are: the client application should use TCP sockets to communicate with the server. Ideally it should be able to uncompress GZIP compressed data received from the server.
5.1.2 Other constraints

It is possible to develop HTML/WML-based applications for the Blackberry environment. It is also possible to write Java applications that rely on HTML. A decision was reached at GNM that any new mobile applications should be written so that they do not rely on HTML or WML, so the client application must be written without using HTML to render the views. The reason for avoiding HTML is that it has been found that a Java application using GZIP compression performs much better than an HTML, browser-based application, when passing large amounts of text data across a limited bandwidth wireless network.

5.1.3 Multi-channel bridging pattern

The multi-channel bridging pattern as stated by Folmer et al. (2006) is reproduced in Appendix G. For the purposes of this study, the most relevant sections of the bridging pattern are ‘Architectural considerations’ and ‘Implementation’, since they provided a standardised approach to compare the three competing design patterns.

From the point of view of the generalised form of a Model-View-Controller design pattern, the model for this application consisted of story text stored in the database. This text would be delivered to the application in the same way, as described in Section 5.1.1, regardless of the design pattern used. Since the server does not hold any state information related to the client, there was no requirement for the model to notify the views of any changes to the underlying data. This applies regardless of the design pattern used. This simplified the design of the application considerably.
The following section shows how each of the design patterns was analysed in the context of this application’s requirements together with the constraints imposed on it by the server.

5.1.4 Comparison of patterns based on architectural and implementation considerations

The three design patterns were compared taking into account architectural and implementation considerations as suggested in the multi-channel bridging pattern.

The modified Service to Worker design pattern fits within the existing server architecture in that a device-independent component similar to the TaskStateBean, called a TaskState, could be implemented. The equivalent of the device-specific ViewStateBean component, called a ViewState, for each view may be required for each device. Since this approach is designed primarily for a Java Server Page/Servlet environment, there is an implicit assumption that the final view will be rendered using HTML. For the project application the TaskState and ViewState components would be implemented as Java objects. Sauter et al. (2005) suggest the use of two XML files to initialise the task and view components at runtime. Implementing their approach would have required changes to the Octopus Content Server. An alternative approach to using configuration files is to store the necessary information in the octopus workflow database. This fits better with the current operation of the Octopus Content Server. It does not do any XML processing, but does have reusable code for retrieving information from an Oracle database. Despite the need for a small change to the Octopus Content Server, the approach fits well within the existing architecture since it allows clear
separation of model and view, and could be deployed without major changes to the server.

The MPCC design pattern also requires new configuration data in the form of AGD and ACD files. In order to make use of these files, the server software would have to be modified. The amount of development required may be acceptable; however this counts against the pattern in this comparison. The controller would be device-specific since it would need to react to user input: the method of input may be different on different devices, for example, track-wheel on a Blackberry whereas a stylus might be used on a Palm device. The coordinator would not be device-specific. The presentations would also be device-specific. An MPCC based design could be used for the project application, but it would also require some changes to the Octopus Content Server. The approach suggested by Hess and Campbell (2003) relies heavily on a middleware component called GaiaOS (Cerqueira et al., 2001). Although it may have been possible to interface the Octopus Content Server with this middleware, this would have required a large amount of effort. This project does not require most of the features that GaiaOS provides. Therefore the benefits do not justify the amount of effort required to implement this design pattern for this project.

Using the Struts framework, an application server or Servlet container, for example the Apache web server combined with Apache Tomcat, would act as a broker between the Octopus Content Server and the client applications. This is because the framework relies on an ActionServlet and Struts taglibs to function. This extra configuration does not require any changes to the Octopus Content Server, since a Java Servlet could be implemented to make the data requests
from the Octopus Content Server, and then provide an API to the Struts components for the data. This does mean that there would be extra server-side configuration (Apache/Tomcat) and development (a Servlet to provide an Octopus Content Server interface). This approach also implies that the data will be rendered using HTML via Java Server Pages (JSP), and since the approach is heavily Servlet/JSP based it would be difficult to create views that did not rely on HTML to be rendered. This is a problem for the project application.

In conclusion, MPCC was discarded since it would have required the GaiaOS middleware to be implemented on the server side. The Struts framework was not used because of its heavy reliance on Servlets and JSP, and because it would have required additional infrastructure changes on the server side. The modified Service to Worker pattern is the fit best within the architectural and implementation constraints of the existing infrastructure at GNM. The modification applied to this design pattern was to remove the reliance of the pattern on Java Beans and XML configuration files.

### 5.2 Implementation

#### 5.2.1 Overview

This section describes the implementation of the application for the Blackberry and MIDP 1.0 environments, using a modified Service to Worker design pattern similar to that suggested by Sauter et al. (2005).

Java™ 2 Platform, Micro Edition, shortened to J2ME, (J2ME platform, 2006) is a common application platform deployed on mobile devices to support applications written in Java. It therefore provides a suitable platform for a heterogeneous
application like the Story Browser. J2ME consists of three main components defined by expert groups consisting of device manufacturers, software vendors and service providers. These components are: the Connected Device Configuration (CDC), Connected Limited Device Configuration (CLDC) and Mobile Information Device Profile (MIDP). CDC is a full-featured Java environment aimed at high-end devices. CLDC is a small footprint configuration aimed at providing a basic Java environment for embedded devices. MIDP together with CLDC and KVM (Sun Microsystems Java virtual machine for mobile devices) provides a runtime environment for mobile phones and PDA’s. It is the MIDP/CLDC environment that the Story Browser application targeted.

One approach to producing heterogeneous applications for mobile devices is to implement it as a MIDlet: a MIDP application derived from the MIDlet class (Pearson publishing, 2006). This provides access to standard UI components supported by MIDP devices. However, this approach would not make best use of any device-specific UI components that may be available. For example, the Blackberry environment provides a number of UI components that build on MIDP to provide significant levels of customisation of properties such as colour and display font. It also provides complex components such as a TreeField component for displaying hierarchies; this is not available in standard MIDP version 1.0 or version 2.0.

The design of the Story Browser allowed it to make use of device-specific API’s where they are available, while using the ‘lowest-common-denominator’ MIDP APIs for devices that do not offer those services. This is important for a user-centred interface since it helps to meet the requirements of user satisfaction and
possible competitive advantage (see Section 3.2). For a particular task within the application, a decision can be made at runtime, based on the configuration stored in the database, about which view, implemented as a Java class, should be shown to the user. On a Blackberry this class was implemented using sophisticated Blackberry API’s to provide a better user-experience, whereas on a less sophisticated mobile phone a MIDlet was deployed that made use of the basic MIDP version 1.0 APIs.

This approach allowed a number of device independent classes to be deployed for areas of functionality not related to the view of data, while using device-specific views to enrich the user experience.

5.2.2 User interface design

The user-interface was well received by users, which suggests that human-centred design techniques proved beneficial for the project. These benefits would apply equally to internal users of the application, and also for customers of GNM if the application were to become a commercial subscription product.

5.2.3 Implementation

The tasks for the application can be considered to be a sequence of tasks. Each of these tasks can be mapped to a single view. The mapping of tasks to view classes for both the Blackberry and MIDP 1.0 implementations is shown in Figure 5.3.

The design called for a device independent TaskState object for each task, with one or more device-specific ViewState objects for each view. As an example, the source code for the classes involved in implementing the book screen view for the
Blackberry and MIDP 1.0 environments are shown in Appendix A. In addition the classes required to implement all views for the MIDP 1.0 application are shown in the same appendix.

The following figures show the screens for StoryBrowser/MidpStoryBrowser, BookScreen/MidpBookScreen, HeadlineScreen/MidpHeadlineScreen and StoryScreen/MidpStoryScreen classes.

The mapping of class to task is:

- **StoryBrowser/MidpStoryBrowser**: choose an issue date. See Figure 5.4
- **BookScreen/MidpBookScreen**: choose a book to view. See Figure 5.5
- **HeadlineScreen/MidpHeadlineScreen**: view headlines. See Figure 5.6 and Figure 5.7
- **StoryScreen/MidpStoryScreen**: view a story. See Figure 5.8
They are shown running on Blackberry, Nokia Symbian S60 and Motorola device emulators. The Blackberry emulator displays the screens for the Blackberry-specific version of the application, while the Nokia and Motorola emulators show the generic MIDlet that supports MIDP 1.0.

Figure 5.4: The initial screen. StoryBrowser and MidpStoryBrowser classes

Differences to note between the Blackberry and MIDlet versions include:

- No coloured text in the MIDlet version (the newspaper mastheads are colour images)

- No control over text formatting in the MIDlet – bold, italic and so on

- No tree widget for the MIDlet to display the section, page and headline hierarchy. Figure 5.6 shows the headline screen with the structure closed, only listing the top level of the hierarchy: a list of sections. The MIDlet is
incapable of showing this. Figure 5.7 shows the list of headlines with page numbers.

**Blackberry**

**Motorola C698p**

**Nokia S60**

Figure 5.5: The book screen. BookScreen and MidpBookScreen classes

Figure 5.6: The headline screen (sections). HeadlineScreen class
Figure 5.7: The headline screen. HeadlineScreen and MidpHeadlineScreen class

Figure 5.8: The story screen. StoryScreen and MidpStoryScreen class
To illustrate how multiple devices can be supported, while making best use of device-specific UI components, consider the implementation of the book screen.

The class diagram for the implementation of the BookScreen view for the Blackberry is shown in Figure 5.9. The screenshot is shown in Figure 5.5.

![Figure 5.9: Simplified class diagram for the Blackberry BookScreen view](image)

The equivalent class diagram for the MIDP 1.0 implementation is shown in Figure 5.10. The differences between the two implementations are highlighted in red text.

![Figure 5.10: Simplified class diagram for the MIDP 1.0 MidpBookScreen view](image)
The implementation of the headline and story views followed the same pattern as this.

The **StoryModel** class is responsible for providing the model from the server. It uses the **ConnectionThread** helper classes to achieve this. The **TaskState** class is capable of instantiating a view object from its class name. The **StoryModel** retrieves the view class names for each task, for an application-device pair, from the server using the “gettaskviewsforappanddevice” message discussed in Section 5.1.1. This information is stored in a **TaskViewClasses** object.

The **TaskState** object uses the **Class.forName()** method to get the object type, and then the **newInstance()** method of the Object class to instantiate an object of that type. In this example a **BookScreen** object is created. This must implement the **ViewState** interface. The **newInstance()** method requires that the target object have a constructor that does not accept parameters. This means that object initialization must be done in a public method. This is the purpose of the **initialize()** method of the **ViewState** object. The **BookScreen** and **MidpBookScreen** objects implement this to initialise any parameters required.

The **ViewState** object is implemented as an interface that must be implemented by view classes. In this case the **BookScreen** class has a realization relationship with the **BlackberryViewState** since it implements that interface. The **initialize()** method has already been discussed. The **BlackberryViewState** extends the **ViewState** interface so that blackberry-specific objects can be passed to the **initialize()** method. The other two
methods in the ViewState interface are startConnection() and endConnection(). The initialize() and startConnection() methods are called by the class that generates the TaskState. See the code for the Dispatcher class for an example of how this is used, and the code for BookScreen to see how a Dispatcher is used. The endConnection() method is a call back method. The ConnectThread executes this method once the data has been retrieved for the server. See the code for StoryModel for an example of this.

The BlackberryConnectThread is a specialisation of ConnectThread. This is so that the UI can display the more user-friendly Blackberry-specific GaugeField class to inform the user of the progress of the data download instead of the standard MIDP Gauge class.

To summarise, the implementation mapped each task to one or more views via a TaskState object that can instantiate a view class using the view class name. The views must implement the ViewState interface. The StoryModel class provides the view class names for each task by querying the database via the Octopus Content Server using the ConnectThread helper class. Therefore new devices can be supported by writing specialised code for their views, and reusing the StoryConstants, TaskViewClasses, Dispatcher, StoryModel, ConnectThread, TaskState and ViewState classes. In the context of the MVC design pattern the classes could be mapped as follows:

- **Model**: StoryModel class with ConnectThread and subclasses, TaskState and TaskViewClasses as helpers
• **View:** BookScreen for Blackberry and MidpBookScreen for MIDP 1.0, with ViewState and subclasses as helpers

• **Controller:** StroyBrowser class with Dispatcher as a helper.

One modification implemented for this project that differed from the approach of Sauter et al. (2005), was that device-specific and view-specific information was collected by the application at runtime from an Oracle database rather from XML configuration files. A new call was added to the Octopus Content Server protocol to support this. This call caused the server to send back device-specific information to initialise the task and view objects as discussed in Section 5.1.1.

The Oracle definitions of the database tables required are shown in Appendix B and Figure 5.11 shows the database table structure. The Oracle JDeveloper tool was used to carry out the database design.

![Database structure for the application-device configuration](image)

The `MOBILE_DEVICES` table stores the list of supported devices. The `MOBILE_APPLICATIONS` table stores a list of supported applications. The

---

Sakib Supple

61 of 121
The **APPLICATION_TASK_STATES** table stores the list of task states for a particular application. The **RUN_ORDER** column of this specifies the sequence of the tasks in ascending order. Finally, the **TASK_VIEW_STATES** lists the views for a particular **APPLICATION_TASK_STATE**. The **RUN_ORDER** column of each row specifies the sequence that the views should be navigated.

The database rows are shown in Appendix B. At the moment only two devices are supported (Table B.1): Blackberry and MIDP1.0. The only application supported is the Story Browser (Table B.2).

There are three states defined for the Story Browser application: Book Screen, Headline Screen and Story Screen. The **RUN_ORDER** is set in ascending order with the first task in the sequence having the lowest value.

For each task in the **APPLICATION_TASK_STATES** table there is one or more entry in the **TASK_VIEW_STATES** table. The **RUN_ORDER** refers to the sequence of navigation for each view per task. Since each task only has one view in this instance, the **RUN_ORDER** values are all set to 1.

### 5.2.4 Mechanism for adding a new device

This section illustrates how Story Browser support for a new device could be added by considering how support for Palm OS mobile devices would be added.

The database configuration tables would have to be updated:

- Add a row to the **MOBILE_DEVICES** table for Palm devices.

- The **MOBILE_APPLICATIONS** table remains unchanged since we are not adding support for a new application.
• Similarly, the `APPLICATION_TASK_STATES` table remains unchanged

• Add rows to the `TASK_VIEW_STATES` table for the Palm-specific view classes for the book, headline and story screens.

The view classes can then be implemented to make best use of the device-specific UI components available. The rest of the existing code can be reused without any changes. Also, if there are no device-specific APIs available for one or more of the screens, the MIDlet view classes could also be used without change.

5.3 Evaluation plan

The user interface evaluation was carried out according to an evaluation plan. This plan follows a similar format to that suggested in M873 (2001, Unit 6, p. 19): explaining the technique to be used, specifying the participants, the tasks to be carried out and the arrangement of the evaluations.

5.3.1 Elements of the evaluation

The performance metrics will be measured by timing critical tasks and analysing the results by comparing them with the non-functional requirements specified in the requirements specification for the application.

The technique used to measure the effectiveness and usability of the user interface will be participatory heuristic inspection (M873, 2001, Unit 6, p. 47; Muller et al., 1998). An important part of this process will be the preparation of a pre-test questionnaire for the evaluation users. This will be used to place their experience and familiarity with mobile devices into context. In addition a form
containing the heuristics for the evaluation will be produced. Evaluation users will use this to report defects in the interface.

5.3.2 Participants

The performance metrics were collected by Sakib Supple who is the developer of the software.

A range of different types of user was chosen for the participatory heuristic evaluation. The participation model for a participatory heuristic evaluation, as defined by Muller et al. (1998), states that inspectors should include:

- Software engineering experts
- Usability and human factors experts
- Work-domain experts

The users chosen were:

- Two Guardian editorial systems staff with extensive experience of user interface design in an editorial environment, as well as experience of the editorial process. They can be thought of as usability and work-domain experts.

- Two Guardian editorial staff, work-domain experts, with a bias towards editorial considerations.

- Two full-time editorial technical support staff, work-domain experts, with a bias towards technology considerations. One of these users was a Blackberry novice.
• Two full-time business systems development staff. These users are software engineering experts.

• Two editorial users who have never used a Blackberry. These users are work-domain experts.

5.3.3 Evaluator

The evaluator was Sakib Supple, who is also the developer of the software. He briefed each participant, ideally face-to-face, about the evaluation tasks and how to fill in the heuristic evaluation form. This included an explanation of the heuristics and the evaluation process.

5.3.4 Evaluation tasks

The tasks for performance testing were identified from the requirement specification (Appendix F) for the application:

• Time taken for book information to be returned to the application

• Time taken for headline information to be returned to the application

• Time taken for story information to be returned to the application

The tasks used during the heuristic evaluation were identified by referring to the requirements specification for the application.

Users will be asked to check the usability of each task against the full set of heuristics.

The tasks chosen for the heuristic evaluation were:

• Choose an issue date for which you would like to view newspaper articles.
• Given a list of newspaper sections, choose one that you would like to view the headlines for.

• Given a list of headlines, choose one that you would like to view the full story text for.

5.3.5 Location of evaluation
The timing of the performance metric tasks were carried out in three locations:

• A training room at The Guardian News & Media offices in Farringdon, London

• My flat in Dulwich, London

• Blackfriars over-ground train station, London

These locations were chosen since they are known to have good and consistent mobile network connectivity.

For the heuristic evaluation, the initial briefings and explanation of the tasks and heuristics took place at the user’s convenience, usually at their desks.

The actual evaluation was carried out at the inspectors’ leisure, while they were away from their desks. That is, in a mobile environment. The evaluators were asked to try and ensure that there was reasonably good network coverage available to their device at the time that the evaluation was carried out.

5.3.6 Running the session and data collection
A heuristic evaluation form was created following the format suggested in M873 (2001, Unit 6, pp. 45-46). This form was supplied in paper format as opposed to
an electronic format, so that the inspectors could carry it with them. This allowed them to carry out the evaluation while on the move, on a train or bus, without the need for a laptop or other device to access the evaluation form.

The form additionally contained a list of the three specific tasks that were to be carried out for the evaluation as a reminder to the inspectors while on the move.

Each inspector was given a list of the heuristics as suggested by Muller et al. (1998). They were also given a verbal explanation of what the heuristics referred to in the context of this application, and the basic theory behind a participatory heuristic evaluation.

The inspectors carried out the evaluation and returned the completed forms to the evaluator. The forms are shown in Appendix C.
5.4 Quantitative analysis

Comparing the raw performance data in Table D.1 of Appendix D to the non-functional requirements in Appendix F, the application failed to meet two of the performance criteria specified, and outperformed for the third:

- Failed: book list retrieved in 10 seconds instead of 4 seconds
- Failed: headline data retrieved in 9 seconds instead of 8 seconds
- Outperformed: story data retrieved in 6 seconds. Within 8 seconds

This is shown graphically in Figure 5.12.

![Figure 5.12: Task timings for the Blackberry story browser](image)

Location of the mobile device did not affect the mean time taken for each task: 10 seconds for retrieving book information, 9 seconds for retrieving headline information and 6 seconds for retrieving story information. This suggests that all three locations had very similar levels of mobile service at the time the tests were performed, or that the application performance is not location-specific. This will need further investigation.
The data transfer rates are shown graphically in Figure 5.13:

![Figure 5.13: Download rates for common tasks](image)

The download rate for the book information, 47 bytes per second, was relatively low. The Octopus Content Server returned the book information as an uncompressed stream, whereas the headline and story information were transferred as compressed data. This may provide a partial explanation of the relatively slow download speed for the book information. The book data was handled differently based on experiments carried out at GNM in early 2005. These suggested that for small amounts of data (typically less than 500 bytes) the overhead of decompressing the data on the Blackberry outweighed the benefits of faster transfer rates due to compression. However, since then the devices in use have improved processing power and memory, so these experiments should be repeated to see if their conclusions are still valid.
Figure 5.14 shows a plot of the amount of data transferred against the mean rate of transfer.

![Figure 5.14](image)

This shows that the rate of data transfer increases as the amount of data increases. One explanation may be that the benefit of transmitting compressed data might increase with the amount of data transferred. This increased benefit may compare favourably with the relative increase in time taken to decompress that data on the device. This requires further investigation, perhaps by increasing the amount of data transferred to see if the transfer rate starts to fall after a certain point. This experiment should be repeated using devices with different hardware characteristics such as CPU and memory. This would show the relationship between transfer rates and hardware characteristics such as CPU and memory.
5.5 Qualitative analysis

The raw data from the qualitative analysis in Table D.2 of Appendix D shows a number of duplicate issues. These duplicates can be removed. Some of the defects reported are a consequence of the nature of the particular device used, or the Blackberry environment in particular. An example of this is the problem of the screen appearing too dim. Similarly some of the problems are a consequence of the user being unfamiliar with the Blackberry environment, for example the problem with the ‘Close’ option in the menu acting as ‘Back’ option rather than closing the menu. The pre-session questionnaire was useful in identifying these since it indicated which users had not previously used a Blackberry. In some cases, the documentation has been criticised for not explaining something, but on investigation the user simply did not read it.

Table D.3 in Appendix D shows a final list with duplicate issues removed, a severity assigned to each problem (1 = minor, 5 = severe) and recommendations given. A useful exercise might be to circulate this table among all the evaluators and then arrange a meeting with them to discuss:

- The accuracy of the final list
- Whether the severity assigned to each defect is reasonable
- Whether the recommendations would resolve the problems identified

The recommendations could then be reviewed and implemented.

5.6 Validation

The final design solution was arrived at using the multi-channel bridging pattern from Folmer et al. (2006). In the context of this project, this method proved to be
less useful than expected. One reason for this is that the interface designer, the software designer and the software engineer were all the same person. Furthermore, that person had intimate knowledge of the server architecture that the application had to function within. This means that the advantage of the bridging pattern method as a communication mechanism between user interface designers and software engineers did not apply in this instance. However, the bridging pattern did provide a set of criteria that proved useful for comparing several design patterns for architectural compatibility with the GNM infrastructure.

The chosen design was a modified version of the Service to Worker design pattern suggested by Sauter et al. (2005). They suggest that their method is useful for breaking down single views into multiple views for devices with differing screen attributes. This project used the method in a different way: to allow a fundamentally heterogeneous application make the most of specialised user interface APIs on more sophisticated devices, while still supporting less sophisticated environments. This is important when using user-centred design techniques to produce a usable user interface.

The resulting user interface was qualitatively evaluated using participatory heuristic inspection. This is a discount method (Ling and Salvendy, 2005) and benefits from being quick, easy and cheap to conduct. Furthermore, some of the inspectors suggested useful solutions to the defects they identified. However, it did suffer from two problems. Some participants found it difficult to understand the heuristics in the context of this application. It may have been better to tailor the heuristics to this application. Secondly, some inexperienced Blackberry users identified features of the Blackberry environment as defects. However this was
mitigated to some extent by identifying these users using a pre-session questionnaire.

The quantitative evaluation of the application identified some performance issues. It also suggested that further investigation is needed on the benefits of data compression for differing quantities of data on different devices.

Although the design called for a small change to the Octopus Content Server, the application did not require any major architectural or infrastructure changes. This constraint was an important requirement of the implementation.

The heterogeneity of the application design was demonstrated by implementing a MIDlet version of the application that could run on any mobile device that supports MIDP 1.0. Although some minor changes were required, the implementation was achieved with relative ease. Also, a simple mechanism was developed to add support for new devices while making best use of device-specific UI components to ensure the best user experience on any particular device.

Therefore it can be seen that the goal of using an MVC-based design pattern to produce a heterogeneous application to distribute newspaper textual content to mobile devices was achieved.
6 Conclusions

This section provides a summary of the research, the results obtained and an indication of future work to be carried out.

6.1 Project review

The project resulted in the implementation of a heterogeneous mobile application to wirelessly publish newspaper text content. This application is suitable for Blackberry devices and other mobile devices that support MIDP 1.0. Furthermore, a clear mechanism exists for adding support for additional mobile devices.

The application was designed using a modified MVC-based pattern similar to that suggested by Sauter et al., (2005). This was chosen by comparing three MVC-based designs using the multi-channel access bridging pattern method suggested by Folmer et al. (2006).

The user interface of the application was designed using user-centred techniques. It was evaluated quantitatively and then qualitatively by a heuristic inspection technique. Although the interface was well-received and considered effective, a number of defects were identified and recommendations made to address them. In order to demonstrate the suitability of the design in producing heterogeneous applications, a more generic MIDlet version of the application was implemented by reusing much of the code used for the Blackberry version.

Although the MIDlet version of the software can run on many devices that support MIDP 1.0, including the Blackberry, the UI is relatively basic and does not necessarily provide the best user experience. The mechanism implemented to
add support for additional devices allows the designer to make the most of the
device-specific UI components that provide the best user-experience, while at the
same time keeping application code changes to a minimum. The Blackberry
application produced here provides proof of this.

In conclusion, the MVC-based, modified Service to Worker design pattern was
successfully applied to the implementation of a heterogeneous application to
distribute newspaper textual content to mobile devices.

6.2 Future work

The qualitative analysis of the Blackberry interface resulted in a series of
recommendations. One useful exercise would be to review these with the
participants of the evaluation and arrive at a final list of recommendations. These
recommendations could then be implemented to repair the defects identified in
the Blackberry user interface. Also the failure of the application to meet the
required performance requirements needs further investigation. In particular the
relationship between data compression and wireless download rates must be
investigated. This requires the investigation of how download rates vary with the
hardware characteristics of the mobile device in use.

The MIDlet version of the user interface must be improved to make better use of
colour and different fonts in the UI. This could be achieved by making all views
subclasses of the Canvas class. Where appropriate, the lessons learned from the
evaluation of the Blackberry interface should be applied to the MIDlet version.
This updated and improved version of the MIDlet interface should then be
evaluated using the participatory heuristic evaluation technique to test its usability.
References


Gamma, E., Helm, R., Johnson R. and Vlissides, J (1997) *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

GSM World (2006), *Mobile 2006* [online]


http://java.sun.com/javame/index.jsp  
[Accessed 14th January 2007].

Jing-Mei Li, Guang-Sheng Ma, Gang Feng and Yu-Qing Ma (2006),  
‘Research on Web Application of Struts Framework Based on MVC  

controller user interface paradigm in Smalltalk-80, Journal of Object-  
26-49.

266-274.

participatory heuristic evaluation’, interactions, Volume 5, Issue 5,  


for Ubiquitous Software’, Proceedings of the 3rd international conference  
on Mobile and ubiquitous multimedia, pp. 71-78.

http://www.developer.com/java/j2me/article.php/10934_1561591_1  
[Accessed 14th January 2007].

component-based adaptable middleware’, Proceedings of the 3rd  

Service to Worker (2002) ‘Core J2EE patterns – Service to Worker’, Sun Microsystems [online]
http://java.sun.com/blueprints/corej2eepatterns/Patterns/ServiceToWorker.html

http://struts.apache.org/1.x/userGuide/introduction.html
[Accessed 26th July 2006].


http://www.idi.ntnu.no/~hal/publications/design-patterns/PLoP2k-Welie.pdf
Accessed 26th March 2006].

http://www.usabilitynet.org/tools/13407stds.htm
[Accessed 30th July 2006].

Glossary

**Cocoon** – A web development framework built using the Java programming language

**Connected Device Configuration (CDC)** – a full-featured Java environment aimed at high-end mobile devices

**Connected Limited Device Configuration (CLDC)** - a small footprint configuration aimed at providing a basic Java environment for embedded devices

**BES** – Blackberry Enterprise Server

**Book** – At GNM this refers to the physical self-contained sections within a newspaper characterised by the fact that they start from page 1 e.g. G2: this is part of The Guardian from Monday to Friday

**GPRS** – General Packet Radio Service. A data transmission technique that transmits and receives information in packets making best use of the radio spectrum.

**GZIP** – An open source compression algorithm

**LAN** – Local area network. A localised network of communicating computers.

**Java™ 2 Platform, Micro Edition (J2ME)** – A common application platform deployed on mobile devices to support applications written in Java

**Mobile Information Device Profile (MIDP)** – Part of the java environment for mobile devices
**MDS** – Mobile Data Services. A Blackberry server process required to provide TCP socket access to mobile applications.

**PDA** - Personal Digital Assistant. Any mobile hand held device that provides computing and information storage retrieval capabilities for personal or business use.

**PL/SQL** – The oracle programming language used to write functions and stored procedures in a database.

**Publication** – At GNM this refers to the overall newspaper product e.g. The Guardian, The Observer.

**Section** – At GNM this refers to the logical section within a book. The main part of the Guardian, referred to internally as ‘G1,’ is split into several sections e.g. Home News, World News, Comment & Debate, Leaders & Reply etc.

**Story** – At GNM this refers to a self-contained article that has a headline, and optionally a byline.

**Symbian OS** – An operating system that is installed on mobile devices and telephones. It is licensed to leading device manufacturers.

**XSLT** – Extensible Style sheet Language Transformation. A language for transforming XML documents into documents of any format (including other XML documents).
Appendix A: Source code for the implementation

The Java source code necessary to understand how the TaskState class and the ViewState interface are used in the Story Browser applications is stored on the attached DVD.

The directory structure is as follows:

- The *Shared* folder contains device-independent classes
- The *Blackberry* folder contains the Blackberry-specific code
- The *MIDlet* folder contains the MIDlet-specific code

Some of the code for the Blackberry version of the application is shown together with all the code for the MIDP 1.0 MIDlet version. The same MIDlet executable ran without any changes in both a Motorola C698p phone emulator, and also a generic Nokia Symbian OS S60 emulator.
Appendix B: Oracle database tables

This appendix shows the scripts to create the additional oracle tables required for the implementation of Story Browser application.

```sql
/*
 * This is the table to hold information about the mobile applications supported
 */
CREATE TABLE MOBILE_APPLICATIONS
(
    ID NUMBER(5) NOT NULL,
    NAME VARCHAR2(64 BYTE) NOT NULL
)

/*
 * This is the table to hold information about the mobile devices supported
 */
CREATE TABLE MOBILE_DEVICES
(
    ID NUMBER(5) NOT NULL,
    NAME VARCHAR2(64 BYTE) NOT NULL
)

/*
 * This is the table to map tasks to an application. The run_order is used to impose an order to the tasks
 */
CREATE TABLE APPLICATION_TASK_STATES
(
    ID NUMBER(10) NOT NULL,
    APPLICATION_ID NUMBER(5) NOT NULL, /* Reference to the ID in MOBILE_APPLICATIONS */
    NAME VARCHAR2(64 BYTE) NOT NULL,
    RUN_ORDER NUMBER(5) DEFAULT 100 NOT NULL
)

/*
 * This maps views to tasks for a particular application and device combination. The run_order would be used if there were multiple views per task. This is currently not used.
 */
CREATE TABLE TASK_VIEW_STATES
(
    ID NUMBER(10) NOT NULL,
    DEVICE_ID NUMBER(5) NOT NULL, /* Reference to the ID in MOBILE_DEVICES */
    TASK_ID NUMBER(10) NOT NULL, /* Reference to the ID in APPLICATION_TASK_STATES */
    CLASS_NAME VARCHAR2(128 BYTE) NOT NULL,
    RUN_ORDER NUMBER(5) DEFAULT 100 NOT NULL
)
```
The following tables show the current contents of the tables in the database. These entries are sufficient to support Blackberry and MIDlet versions of the application.

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Blackberry</td>
</tr>
<tr>
<td>2</td>
<td>MIDP1.0</td>
</tr>
</tbody>
</table>

Table B.1: Contents of the MOBILE_DEVICES table

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Story Browser</td>
</tr>
</tbody>
</table>

Table B.2: Contents of the MOBILE_APPLICATIONS table

<table>
<thead>
<tr>
<th>ID</th>
<th>APPLICATION_ID</th>
<th>NAME</th>
<th>RUN_ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Book Screen</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Headline Screen</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>Story Screen</td>
<td>3</td>
</tr>
</tbody>
</table>

Table B.3: Contents of the APPLICATION_TASK_STATES table

<table>
<thead>
<tr>
<th>ID</th>
<th>DEVICE_ID</th>
<th>TASK_ID</th>
<th>CLASS_NAME</th>
<th>RUN_ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>StoryBrowser.BookScreen</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>StoryBrowser.HeadlineScreen</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>StoryBrowser.StoryScreen</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>StoryBrowser.MidpBookScreen</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2</td>
<td>StoryBrowser.MidpHeadlineScreen</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>3</td>
<td>StoryBrowser.MidpStoryScreen</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.4: Contents of the TASK_VIEW_STATES table
Appendix C: Usability evaluation forms

Three evaluation documents were produced: pre-session questionnaire, an explanation of participatory heuristic evaluation and an evaluation form. Hard copies of these were given to each evaluator.

Pre-session questionnaire

Please answer the following questions before carrying out the evaluation of the software. These will be used to evaluate your experience of using mobile devices. Your name will not be stored with the information that you provide.

<table>
<thead>
<tr>
<th>Question</th>
<th>Yes</th>
<th>No</th>
<th>Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do you have a Blackberry?</td>
<td></td>
<td></td>
<td>If yes, how long have you been a Blackberry user? years.</td>
</tr>
<tr>
<td>Do you have a mobile phone?</td>
<td></td>
<td></td>
<td>If yes, how long have you been a mobile phone user? years.</td>
</tr>
<tr>
<td>Do you have another mobile device or PDA?</td>
<td></td>
<td></td>
<td>If yes, how long have you used this for? years.</td>
</tr>
</tbody>
</table>

Table C.1: The pre-session questionnaire

Participatory Heuristic Evaluation

There follows a list of usability guidelines that a ‘good’ user interface should follow. Each guideline is given a number and this can be used as shorthand to refer to it during the evaluation. The accompanying evaluation forms can be used to report
any defects i.e. aspects of the interface that do not follow these guidelines. Please complete the form as directed by the evaluator.

**System Status**

1 SYSTEM STATUS

The system keeps users informed about what is going on through appropriate feedback within a reasonable time.

**User Control and Freedom**

2 TASK SEQUENCING

Users can select and sequence tasks (when appropriate), rather than the system taking control of the users’ actions. Wizards are available but are optional and under user control.

3 EMERGENCY EXITS

Users can:

- Easily find “emergency exits” if they choose system functions by mistake (emergency exits allow the user to leave the unwanted state without having to go through an extended dialogue).
- Make their own decisions (with clear information and feedback) regarding the costs of exiting current work.
- Access undo and redo operations.

4 FLEXIBILITY AND EFFICIENCY OF USE

Accelerators are available to experts, but are unseen by the novice. Users are able to tailor frequent actions. Alternative means of access and operation are available for users who differ from the “average” user (e.g., in physical or cognitive ability, culture, language, etc.).

**Consistency and Relevancy**

5 MATCH BETWEEN SYSTEM AND THE REAL WORLD
The system speaks the users’ language, with words, phrases, and concepts familiar to the user, rather than system-oriented terms. Messages are based on the users’ real world, making information appear in a natural and logical order.

6 CONSISTENCY AND STANDARDS

Each word, phrase, or image in the design is used consistently, with a single meaning. Each interface object or computer operation is always referred to using the same consistent word, phrase, or image. Follow the conventions of the delivery system or platform.

7 RECOGNITION RATHER THAN RECALL

Objects, actions, and options are visible. The user does not have to remember information from one part of the dialogue to another. Instructions for use of the system are visible or easily retrievable whenever appropriate.

8 AESTHETIC AND MINIMALIST DESIGN

Dialogs do not contain information that is irrelevant or rarely needed (extra information in a dialog competes with the relevant units of information and diminishes their relative visibility).

9 HELP AND DOCUMENTATION

The system is intuitive and can be used for the most common tasks without documentation. Where needed, documentation is easy to search, supports a user task, lists concrete steps to be carried out, and is sized appropriately to the users’ task. Large documents are supplemented with multiple means of finding their contents (tables of contents, indexes, searches, etc.).

Error Recognition and Recovery

10 HELP USERS RECOGNIZE, DIAGNOSE, AND RECOVER FROM ERRORS

Error messages precisely indicate the problem and constructively suggest a solution. They are expressed in plain (users’) language (no codes). Users are not blamed for the error.
11 ERROR PREVENTION

Even better than good error messages is a careful design that prevents a problem from occurring in
the first place. Users’ “errors” are anticipated, and the system treats the “error” as either a valid
input or an ambiguous input to be clarified.

Task and Work Support

12 SKILLS

The system supports, extends, supplements, or enhances the user’s skills, background knowledge,
and expertise. The system does not replace them. Wizards support, extend, or execute decisions
made by users.

13 PLEASURABLE AND RESPECTFUL INTERACTION WITH THE USER

The user’s interactions with the system enhance the quality of her or his experience. The user is
treated with respect. The design reflects the user’s professional role, personal identity, or intention.
The design is aesthetically pleasing - with an appropriate balance of artistic as well as functional
value.

14 QUALITY WORK

The system supports the user in delivering quality work to her or his clients (if appropriate).
Attributes of quality work include timeliness, accuracy, aesthetic appeal, and appropriate levels of
completeness.

15 PRIVACY

The system helps the user to protect personal or private information - belonging to the user or to his
or clients.
<table>
<thead>
<tr>
<th>Task description no.</th>
<th>Session date:</th>
<th>Session start time:</th>
<th>Session end time:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluator: Sakib Supple</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inspector:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Location in the task description</td>
<td>Requirements(s) violated</td>
<td>Usability defect description</td>
<td>Additional comments regarding the usability defect</td>
</tr>
</tbody>
</table>

**Figure C.1: The evaluation form**

**TASK 1:** Choose an issue date for which you would like to view newspaper articles. Then request the books published for that issue date.

**TASK 2:** Given a list of newspaper books, choose one that you would like to view the headlines for, and request the headlines for it.

**TASK 3:** Given a list of headlines, choose one that you would like to view the full story for, and request the full story.
Appendix D: Raw evaluation data

Table D.1 shows the raw Performance test results collected for the Blackberry application.

<table>
<thead>
<tr>
<th>Location/Task</th>
<th>Time taken to the nearest second</th>
<th>Mean time/Secs</th>
<th>Bytes</th>
<th>Bytes/Second</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Location 1</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Book information</td>
<td>14 10 10 9 13 9 9 10 9 9</td>
<td>10 465</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>Headline information</td>
<td>10 9 9 9 10 9 9 9 9 9</td>
<td>9 15760</td>
<td>1751</td>
<td></td>
</tr>
<tr>
<td>Story information</td>
<td>6 6 5 6 6 5 6 6 6 6</td>
<td>6 4160</td>
<td>693</td>
<td></td>
</tr>
<tr>
<td><strong>Location 2</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Book information</td>
<td>12 11 9 10 10 9 9 10 9 9</td>
<td>10 465</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>Headline information</td>
<td>11 9 9 9 9 9 9 9 9 9</td>
<td>9 15760</td>
<td>1751</td>
<td></td>
</tr>
<tr>
<td>Story information</td>
<td>5 5 6 6 6 5 6 6 6 6</td>
<td>6 4160</td>
<td>693</td>
<td></td>
</tr>
<tr>
<td><strong>Location 3</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Book information</td>
<td>10 10 12 9 9 9 9 10 9 10 9</td>
<td>10 465</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>Headline information</td>
<td>10 9 9 9 10 9 9 9 9 9</td>
<td>9 15760</td>
<td>1751</td>
<td></td>
</tr>
<tr>
<td>Story information</td>
<td>6 6 5 6 6 5 6 6 6 6</td>
<td>6 4160</td>
<td>693</td>
<td></td>
</tr>
</tbody>
</table>

Table D.1: Performance testing for common tasks
Table D.2 shows the raw evaluation data collected during the heuristic evaluation of the Blackberry application.

<table>
<thead>
<tr>
<th>Task</th>
<th>Location in task</th>
<th>Heuristic</th>
<th>Defect</th>
<th>Comments</th>
<th>Years of experience of mobile devices</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Blackberry</td>
</tr>
<tr>
<td>1</td>
<td>Waiting for response</td>
<td>1. System status</td>
<td>Percentage displayed is not proportionate to time</td>
<td>Pause at zero percent suggests process has stalled</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>As above</td>
<td>1. System status</td>
<td>As above</td>
<td>Minor</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>Status screen after selecting data</td>
<td>1. System status</td>
<td>If reception lost, error does not seem to appear</td>
<td>Perhaps the timeout period for getting info is too long</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Waiting for response</td>
<td>1. System status</td>
<td>Used to seeing an egg timer. Didn't notice the progress bar</td>
<td>Would prefer an egg timer in center of screen</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>Getting books after setting date</td>
<td>4. Flexibility and efficiency of use</td>
<td>Hitting return should get books. Instead forced to use menu</td>
<td>User would expect return to get books</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Selecting the menu option</td>
<td>5. Match between system and real world</td>
<td>Books is a term the user may not be familiar with in this context</td>
<td>Minor</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>Selecting the menu option</td>
<td>5. Match between system and real world</td>
<td>I would prefer to see the paper split into sections, not books</td>
<td>Offset by the fact that the book is split into sections</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>Opened menu to change date</td>
<td>7. Recognition rather than recall</td>
<td>Function of 'Close' option in menu not clear. Thought it meant 'close the menu. Actually closes the application</td>
<td>-</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>Reading text</td>
<td>8. Aesthetics and minimalist design</td>
<td>Blue text on white background difficult to read</td>
<td>Change colours</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>Readability of screen title</td>
<td>8. Aesthetics and minimalist design</td>
<td>Blue text on red is the theme for my blackberry. This makes the title bar difficult to read</td>
<td>Perhaps the application should always use a particular colour, overriding the theme</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Application icon</td>
<td>8. Aesthetics and minimalist design</td>
<td>The application icon not clear</td>
<td>Change colours to make icon clearer</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>Alt key specified in doc, but no key marked 'Alt'</td>
<td>9. Help and documentation</td>
<td></td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>Documentation headings</td>
<td>9. Help and documentation</td>
<td>Headings are not user-centered</td>
<td>Could focus on what the user wants to do rather than what the device can do</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>Changing date</td>
<td>9. Help and documentation</td>
<td>Took a while to work out change date by scrollwheel</td>
<td>Make the doc clearer</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Presumption in the doc that a user would know what ALT key is</td>
<td>9. Help and documentation</td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>---</td>
<td>-------------------------------------------------------------</td>
<td>---------------------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>Waiting for response</td>
<td>1. System status</td>
<td>Progress bar does not change although can see comms activity</td>
<td>Interface otherwise clear</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>Waiting for response</td>
<td>1. System status</td>
<td>Percentage displayed is not proportionate to time</td>
<td>Pause at zero percent suggests process has stalled</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Getting headlines</td>
<td>6. Consistency and standards</td>
<td>Get headlines actually gets sections</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>General</td>
<td>8. Aesthetics and minimalist design</td>
<td>Screen is a bit dim.</td>
<td>Possible to make screen brighter</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>Getting headlines</td>
<td>9. Help and documentation</td>
<td>Get headlines actually gets sections</td>
<td>Get sections’ menu option might be clearer</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>Waiting for response</td>
<td>1. System status</td>
<td>Percentage displayed is not proportionate to time</td>
<td>Pause at zero percent suggests process has stalled</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Expanding and collapsing hierarchy</td>
<td>4. Flexibility and efficiency of use</td>
<td>Would expect +/- keys to expand and collapse hierarchy</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Selecting a headline</td>
<td>4. Flexibility and efficiency of use</td>
<td>Extra prefixes in headline can make selecting difficult</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Displaying a story</td>
<td>8. Aesthetics and minimalist design</td>
<td>Story links to other pages not supported</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Selecting a story</td>
<td>8. Aesthetics and minimalist design</td>
<td>Not clear which bit of the hierarchy refers to a story</td>
<td>Use different fonts to differentiate between stories and other info</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Choosing a headline</td>
<td>9. Help and documentation</td>
<td>The edition notation is not explained</td>
<td>Not sure all users would understand about editions</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Choosing a headline</td>
<td>13. Pleasurable and respectful interaction with the user</td>
<td>Difficult to read long headlines since the text does not wrap</td>
<td>You can’t be sure what the story is about unless you can read the whole text</td>
<td>-</td>
</tr>
</tbody>
</table>

Table D.2: The raw evaluation data for the Blackberry application
Table D.3 shows a final list with duplicate issues removed, a severity assigned to each problem (1 = minor, 5 = severe) and recommendations given.

<table>
<thead>
<tr>
<th>Task</th>
<th>Heuristic</th>
<th>Defect</th>
<th>Severity</th>
<th>Recommendations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,3</td>
<td>1. System status</td>
<td>Progress bar is inaccurate</td>
<td>4</td>
<td>Pause at zero percent suggests process has stalled</td>
</tr>
<tr>
<td>1</td>
<td>4. Flexibility and efficiency of use</td>
<td>Hitting return should get books. Instead forced to use menu</td>
<td>2</td>
<td>Change the interface so that hitting return will get the book list</td>
</tr>
<tr>
<td>1</td>
<td>5. Match between system and real world</td>
<td>Books is a term the user may not be familiar with in this context</td>
<td>1</td>
<td>For internal user this is not an issue. If the product is offered to the public, this will need to be looked at.</td>
</tr>
<tr>
<td>1</td>
<td>8. Aesthetics and minimalist design</td>
<td>Blue text on white background difficult to read</td>
<td>3</td>
<td>Allow the user to set the text foreground and background colour as a persistent application preference.</td>
</tr>
<tr>
<td>1</td>
<td>8. Aesthetics and minimalist design</td>
<td>Blue text on red is the theme for my blackberry. This makes the title bar difficult to read</td>
<td>1</td>
<td>Allow the user to set the title colour as a persistent application preference.</td>
</tr>
<tr>
<td>1</td>
<td>9. Help and documentation</td>
<td>Headings are not user-centered</td>
<td>3</td>
<td>Provide user-centred documentation</td>
</tr>
<tr>
<td>2</td>
<td>6. Consistency and standards</td>
<td>Get headlines actually gets sections</td>
<td>4</td>
<td>Change the menu option to more accurately reflect what is happening</td>
</tr>
<tr>
<td>3</td>
<td>4. Flexibility and efficiency of use</td>
<td>Would expect +/- keys to expand and collapse hierarchy</td>
<td>2</td>
<td>Canvas opinion amongst other evaluators to see if they agree with this.</td>
</tr>
<tr>
<td>3</td>
<td>4. Flexibility and efficiency of use</td>
<td>Extra prefixes in headline can make selecting difficult</td>
<td>3</td>
<td>Make the headline text wrap, so that the prefix and all the headlines can be read.</td>
</tr>
<tr>
<td>3</td>
<td>8. Aesthetics and minimalist design</td>
<td>Story links to other pages not supported</td>
<td>1</td>
<td>Investigate implementing links to other stories.</td>
</tr>
<tr>
<td>3</td>
<td>8. Aesthetics and minimalist design</td>
<td>Not clear which bit of the hierarchy refers to a story</td>
<td>2</td>
<td>Use different fonts and formatting to differentiate between stories and other information on the screen</td>
</tr>
<tr>
<td>3</td>
<td>9. Help and documentation</td>
<td>The edition notation is not explained</td>
<td>1</td>
<td>For internal user this is not an issue. If the product is offered to the public, this will need to be looked at.</td>
</tr>
</tbody>
</table>

Table D.3: Final evaluation data
Appendix E: Objectives, ethics, resources and risks

This appendix shows the sections on objectives, ethics, resources and risks that have been removed from the main body of the dissertation.

Objectives

Objective 1: To investigate, using the literature available, the current use of MVC-based design patterns for producing heterogeneous mobile applications.

Tasks for objective 1:

a) Re-write the existing requirement specification to make it suitable for this project. It will need simplifying. This is necessary to put the rest of the work into context.

b) Obtain a copy of ‘Design patterns: Elements of reusable object-oriented software’ [7] and make notes on the sections related to MVC

c) Use current literature search materials to build a literature search list

d) Use the library links to web bibliographies on the course web site to find more research on MVC

e) Make clear notes on the MVC-based design patterns found that are appropriate for mobile applications

f) Use the notes for d) and e) to generate MVC-based designs

Methods:
a) For tasks c and d: use keyword searches for ubiquitous, pervasive, model-view-controller, MVC, MPCC, Struts, Service to Worker

**Deliverable for objective 1:**

A list of three MVC-based designs to be evaluated and compared in objective 3

**Milestones for objective 1:**

Requirements specification complete and a list of three MVC-based designs patterns currently used for heterogeneous mobile applications by the 5th of August 2006

**TMA03 – 15th August**

**Objective 2:** To analyse and evaluate the competing MVC-based design patterns (delivered by objective 1) to ascertain the most appropriate pattern for the story browser application, and to ascertain the effectiveness of the bridging pattern method as an evaluation tool for this purpose

**Tasks for objective 2:**

a) Compare the design patterns for compatibility with the existing GNM server infrastructure

b) Compare the design patterns for ability to deliver heterogeneous applications

**Methods:**
a) For tasks a and b: use the multi-channel bridging pattern [3] to compare the different design patterns

**Deliverable for objective 2:**

The most appropriate MVC-based design pattern for the story browser application.

**Milestone for objective 2:**

The most appropriate MVC-based design pattern for the story browser application by 20th August 2006

**TMA04 – 14th November**

**Objective 3:** To evaluate the effectiveness, or otherwise, of the design pattern chosen in producing usable mobile applications

**Tasks for objective 3:**

a) Create a design for the application based on the chosen design pattern delivered by objective 2

b) Use the Blackberry development environment to implement

c) Unit and integration test the application

d) Provide user documentation for the application

e) Liaise with the GNM Blackberry systems administrators to deliver the application to the test users
f) Define usability metrics for the application (the requirements specification may include non-functional requirements that will help with this)

g) Define an evaluation plan to include

   a. Elements of the evaluation

   b. Participants in the evaluation

   c. Tasks to be carried out

   d. Where and how the evaluation is to be carried out

h) Execute the evaluation plan

   a. Carry out usability metric tests

   b. Carry out participatory heuristic evaluation

i) Analyse results of the evaluation

j) Reach conclusions from the evaluation

Methods:

a) For task a: use UI design methods defined in the M873 OU course and the ISO 13407 standard

b) For task b: Java network programming

c) For tasks c, d, e and f: use methods defined in the M873 OU course and by Muller et al
Deliverable for objective 3:

UI evaluation results and conclusions

Milestone for objective 3:

UI evaluation results and conclusions by 3rd December 2006

Objective 4: To demonstrate the ability, or otherwise, of the chosen design pattern to produce heterogeneous mobile applications

Tasks for objective 4:

a) Implement the design (delivered by objective 3) on a mobile phone or desktop environment, noting how much extra work had to be done over and above that required for the application delivered by objective 4

b) Unit and integration test the application

c) Reach a conclusion as to how good the design pattern was at producing a heterogeneous application

Methods:

a) For task a) use Java network programming

Deliverable for objective 4:

Conclusion on heterogeneity

Milestone for objective 4:

Conclusion on heterogeneity by 15th December 2006
Risk assessment

This risk assessment has been reviewed for TMA03 and the risks, and the steps taken to mitigate them, remain unchanged. There have been no serious problems and I am still confident that the project can be completed on time.

This risk assessment has been carried out using the approach suggested in M865 (2004, Unit 2, pp. 35-40) dealing with the high-impact, high-probability risks first, then the high-impact, lower-probability risks and finally the lower-impact, lower-probability risks.

In general the following steps have already been taken to avoid risks:

- There is no dependency on external contractors for hardware or software supply
- There is no dependency on external technical resources for design or coding
- All software and hardware requirements have already been identified and sourced
- I already possess the technical training and experience required for the project so training is not required
- A willing group of users has already been identified for the user evaluation

- GNM have agreed to bear any costs for additional resources for the project, so the risk of increased budget requirements has been transferred to GNM

The remaining risks are shown below.

**High-impact, high-probability risks**

- **Test users become unavailable.** The group of test users I have agreements from to help with the project could become unavailable for reasons of illness or workload.

- The risk of workload problems has been reduced by recruiting users from different departments in GNM – a specific large-scale project in one area will have a reduced impact on my project

- The risk of illness can be dealt with by recruiting more users. GNM has over 200 Blackberry users some of whom have shown a willingness to help with this project. Mobile wireless access to our content currently has a high profile within the organisation so many people are interested in getting involved for that reason alone.

**High-impact, lower-probability risks**

- **Serious illness or injury.** If I suffer serious illness or injury that requires one month or more of rest, then realistically the project will have to be abandoned (risk avoidance). The schedule is fairly tight and it would be difficult to recover from this situation.
• **Increase in my workload.** All of my work at GNM is project-based. I have a clear idea of my project schedules with objectives having been set for the 12 months between April 2006 and April 2007. As things stand I have been able to set aside time at work specifically for this project – it is one of my official objectives for the year. However, priorities may change so that one of my other projects needs to take precedence (or new business requirements give rise to a new project). To mitigate this I have built in a 10% contingency in my time at work set aside for this project. This situation will be monitored carefully as the project progresses.

• **Not enough resources to code the application.** It may be that the initial design evaluation and analysis takes longer than expected. In that case I may not have enough time to code the application by myself. I have been given the authority to recruit extra programming resources (ideally from within GNM, but externally if that is not possible) to help me with this. It would be wise to monitor the situation carefully and report back to my supervisor any slippage in the analysis work.

**Lower-impact, lower-probability risks**

• **Not all software and hardware requirements identified.** Some extra software or hardware may additionally be required. I have agreed the authority to source any additional equipment through GNM.

**Resources**

I have reviewed the resource requirements (for TMA03) and the only change is that I now have eight Blackberry users who have agreed to take part in the
evaluation. I am actively engaged in trying to recruit two more so that I will have a
total of ten users to carry out the evaluation with.

**Hardware:**

- Any Java development and documentation will be done on a Macintosh PC
  (my preferred platform). This is supplied by GNM.

- Windows PC to run the blackberry development environment and
  simulators. This is supplied by GNM.

- A blackberry mobile device, and perhaps a Java-enabled mobile phone
  with mobile subscriptions. GNM has already supplied both of these.

- A Solaris computer to run the server software, and access to the Guardian
  and Observer text archive (an oracle database). Again this has been
  supplied.

**Software:**

- Project management software to produce a project plan, Gantt charts and
  task breakdowns. I have a licensed copy of FastTrack Schedule 8 supplied
  by GNM.

- Drawing tools to produce any diagrams (UML, UI representations etc.). I
  have licensed copies of Adobe Illustrator and OmniGraffle Professional. I
  have successfully used both of these in the past to produce different types
  of technical diagrams.

- Microsoft Office to produce documents and spreadsheets. Licensed copies
  are provided by GNM.
Books:

- Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gamma et al. This is on order.

People and other:

- Access to a small group of users (internal to the organisation) for UI evaluation. Eight people have already agreed to help with the testing and evaluation of the application. I am actively recruiting two more users to make a total of 10. If this is still insufficient, the application can, if required, be deployed to over 200 Blackberry users at GNM. I will therefore have a large pool of users to canvas for support if I need to recruit more.

- Tools for analysis of any evaluation metrics. I do not anticipate needing anything more than a calculator for this, and I have access to a Casio calculator with statistical functions.

- Access to, and permission to schedule work for, the Blackberry administrators at GNM. This has been agreed with my employer.

- Internet access from a PC/Mac for accessing OU library and other resources (libraries, online databases, citation indices). I have constant access to the Internet both at home and at work.

- Time to do the software development work. This has been agreed with my employer.
• Any programming will require the following language skills: Java, PL/SQL. I already possess these to a reasonable standard so no further training is required.

• Access to additional programming resources if required. I have the authority to recruit more resource if needed.

• Reference and tutorial documentation for the Blackberry API, CLDC/MIDP and J2ME. These are available online and have been downloaded and found to be adequate.

**Ethical issues**

There are no formal ethical guidelines aimed at IT professionals at GNM. However the organisation does have to operate within certain legal constraints, and in addition employees are expected to aspire to a certain level of professional conduct. This section will provide a set of formal ethical and legal guidelines for this particular project, and these will be based on the existing framework within which projects at GNM are already carried out.

Much of the work at GNM is carried out within constraints that are similar to those imposed by the British Computer Society (BCS) code of conduct [4] and the Association of Computing Machinery (ACM) code of conduct [5]. The following set of guidelines could be applied to this project:

• Copyrights in relation to editorial (text and pictures) content must be honoured
• Copyrights and the terms of license agreements in relation to software and hardware usage must be honoured [4,5]

• Project team members must operate within existing local, national and international legal frameworks. This requires that the project team must be aware of and understand all relevant legislation, regulations and standards [4, 5]. For example the Data Protection Act: in the first instance no personal data will be collected by the system so this should not be an issue; however if the system is adapted to provide a subscription service to readers, then customer data will be collected and must be protected

• The privacy of participants and stakeholders in the project must be respected [4]. This applies to all participants in the evaluation process, and precautions must be taken to ensure the protection of all data collected during the project. This data will only made more freely available with the participants express permission.

• All assessments and claims made about the system created during the project must be accurate and truthful [5].

• Matters of health and safety must be considered during the course of the project [4,5]

As an example of how these guidelines could be applied during the project, consider the issue of health and safety.

The UK has the longest working hours in Europe. The work-life balance achieved by employees has been an area of concern for some time in the business world,
with a number of health and social risks being associated with long working hours [1,2,3].

The BCS code of conduct states that “in your professional role you shall have regard for the public health, safety and environment” [4]; the ACM code of conduct states that one should ensure that one’s work “will be used in socially responsible ways, will meet social needs, and will avoid harmful effects to health and welfare” [5]. By allowing mobile access to corporate data as this project aims to do, it may seem that my approach will fall foul of these codes of conduct: it allows out-of-hours access to information in such a way that may encourage an increase in working hours. However, there is an alternative view of this.

There is a downward trend in the number of hours worked per week in the UK, partly due to legislation such as the EC Working Time Directive [2,3] and partly due to the increase in flexible working [2]. By allowing mobile access to corporate data, it can be argued that employees are encouraged to work in more flexible ways thus improving their overall work-life balance while not increasing their working hours. For example a parent may take their child to the park to play with friends, and still continue to work at the park using mobile access; the environment is more pleasant than an office and the parent and child are spending more time together.

Furthermore, there is evidence that mobile access allows employees to make use of ‘dead’ time (commuting for example) to carry out work that reduces pressure on them when they arrive at the office [6].
In addition to the points made above, due to the specialised nature of the editorial process there are a number of specific legal issues that need further discussion. The content (articles and pictures) published by GNM is subject to copyright legislation: Copyright, Designs and Patents Act 1988. Individual contributors to The Guardian and Observer newspapers have different contracts with the organisation, for example:

- The copyright of text and pictures produced by full time employees belongs to GNM for the purposes of print and internet publishing

- Freelance writers and photographers tend to sign contracts with clauses that pass the copyright of their work to GNM for the purposes of print and internet publishing

- Certain contributors have special contracts that ensure that the copyright remains with them for all content they generate (for example the cartoonist Steve Bell). In this case systems must be put in place to ensure that this copyright is respected. This is particularly relevant for this application since it involves the reuse of content initially produced for print publishing. The Octopus Content Server interfaces directly to the GNM Rights and Contributors System (RCS) to ensure that copyrights are not infringed. The Octopus Content Server cannot access content for unless specific syndication rights have been granted to allow it.

Since the application will initially only be used internally for research purposes by GNM employees, none of these issues apply. They would become important if the application was adapted to provide a service to readers.
No specific ethical issues were raised during this project.

**References for Appendix E**

http://www.cipd.co.uk/subjects/wrkgtime/wrktmewrklfbal/worklifeba.htm
[Accessed 29th May 2006]

[2] Chartered Institute of Personnel and Development (n.d.), *Working hours in the UK* [online]
http://www.cipd.co.uk/subjects/wrkgtime/general/ukworkhrs.htm
[Accessed 29th May 2006]

[Accessed 29th May 2006]

http://www.bcs.org/server.php?show=conWebDoc.1588
[Accessed 29th May 2006]

http://www.acm.org/constitution/code.html
[Accessed 29th May 2006]


Appendix F: Requirements specification for the story browser

A simplified requirements specification has been produced for the purposes of this project. This is one of the deliverables for the project.

Document Overview

This document will provide a list of user requirements for the Octopus Story Browse.

The story browser will provide access to the editorial text content of all publications published by Guardian Newspapers Limited.

It will cover:

- The functional user requirements for the story browser
- The non-functional user requirements for the story browser

Functional Requirements

This is a list of business functionality that the client application must provide.

Issue date

The user will be able to select an issue date to view the editorial text for.

- It will be possible to independently change the day, month and year of the issue date
- Only valid dates will be allowed (taking into account month and leap years)
- Dates in the future and the past will be allowed
- The default date will be the tomorrow

Books

For the selected issue date, it will be possible to list the books for which editorial content exists at the time of the request.

- It will be possible to select one of these books to request the headlines of all the stories published within it
Headline information

The information about the headlines in a book (as requested above) will be presented as a section-page-headline hierarchy

- The book will be represented as consisting of one or more sections e.g. Home News, Foreign News etc
- Each section will be represented as consisting of one or more pages e.g. Home News is made up of pages one to eight
- Each page will list the headlines for each story that starts on that page
- It will be possible to select a headline from the list to request the full text of the story

Story information

The story will be displayed so that the user can read the full content

- The headline, byline (if one exists) and the body text of the story will be displayed
- It will be clear which portion of the text is the headline, which portion is the byline and which is the text
- It will be possible to navigate quickly to the next or previous story
- It will be possible copy the whole story to the clipboard in a structured format i.e. the headline followed by the byline and then the body text, so that it can be pasted into other applications on the device.

Non-functional Requirements

This section lists requirements that the story browser must be meet that do not provide a specific piece of business functionality.

Performance

The information should be returned to the requestor within the following time frames:

- Book list: within 4 seconds
- Headline information: within 8 seconds
- Story information: within 8 seconds
Usability

The interaction between the user and the story browser should meet the following general requirements:

- Visibility
- Affordability
- Feedback
- Natural mapping
Appendix G: Multi-channel bridging pattern as specified by Folmer et al

This appendix reproduces the multi-channel bridging pattern as stated by Folmer et al [3].

Multi-channel access

1. Problem

A user wants or requires access to a system using different devices (mobile phone, desktop, PDA).

2. Use when

You are designing a web-based system (such as an e-commerce system) that targets many users. To increase the number of potential users/customers the accessibility of the system is increased by supporting multiple devices.

The need for a device is either determined by the users (for example a disabled person with speech input device) or either by the user context (for example a mobile phone) or by a combination of these.

An application that has been designed for the Web will contain lists of items that would fit in a desktop screen without problems but that we are unable to display completely on the very small screen of a mobile phone. Interaction on a mobile phone is also limited.
Therefore, the same software functionality is required to be manipulated and presented using different human–computer controls and different interface styles for different user preferences, needs or disabilities.

Devices are different with regard to input device (e.g. mouse, stylus, keyboard, voice) but also with regard to output device (screen size, resolution, screen colors, audio, etc.). In addition, devices may pose constraints because they are limited concerning hardware (memory, CPU power), Operating systems (multi/single threaded), communication types (asynchronous/synchronous) and software (languages/character sets/browsers) posing constraints on the ability to present functionality in the same way as for example on a desktop computer.

3. Solution

3.1. Provide a mechanism that provides multiple channels that are specialized to support different devices

Multi-channelling defines the user’s accessibility to a system through more than one channel. Each channel specifies a certain presentation and/or control set for a device or group of devices. Different input devices often also prescribe certain interface components (such as projecting a keyboard on a touch screen). How this information is presented and manipulated may be different for each device but the accessibility (e.g. access to the system) is the same for each user.

4. Why

Accessibility may increase satisfaction by allowing the use of the system adapted to the users (user context, device, disability, etc.).
Providing device-specific views may aid visual consistency and functional consistency hence contributing to learnability.

Having device-specific views available at any time will contribute to error prevention and minimize cognitive load.

5. More examples

Ebay.com can be accessed through conventional channels such as a desktop computer, but also through more advanced channels such as an I-mode/WAP phone or a PDA. Although the how the information is presented and manipulated is different for each device, the accessibility is the same for each channel which allows users to fulfill their goal of use (e.g. buy a specific object on eBay).

6. Architectural considerations

The architecture must be constructed in such a way that new channels can be easily added and maintained. There are several architecture styles [14] that fulfill these responsibilities.

6.1. 2-tier architecture

The classical 2-tier architecture divides the software system into server and clients. The client application contains business logic as well as the complete interaction. There is no separation of functionality and interaction. In traditional monolithic/2-tier applications when adding new channels business functionality and interaction logic needs to be duplicated for each channel. This solution works fine if the number of devices that need to be supported is small.
6.2. 3-tier architecture

A more elegant solution is to use a 3-tier architecture. The 3-tier architecture separates the application into two tiers by separating the application-logic from the user interface. In most cases, the application-logic part of the software on the server-side is specially designed for a specific type of user interface. In this case the presentation is encapsulated but the interaction is not separated from business logic. When adding new channels part of the business logic and interaction in the second layers need to be duplicated. 2- and 3-tiered architecture are all examples of the architectural pattern layers [13].

6.3. Model view controller

One step further is to decouple the interaction from the business logic. The MVC pattern [14] is a way of breaking an application, or even just a piece of an application’s interface, into three parts: the model, the view, and the controller.

- The view manages the output device (PDA screen, voice) that is allocated to its application.

- The controller interprets input device (mouse, stylus or keyboard) from the user, commanding the model and/or the view to change as appropriate.

- Finally, the model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller). The model component needs to notify the view component when the model is updated, so that the display can be redrawn.
This pattern decouples changes to how data are manipulated from how they are displayed or stored, while unifying the code in each component. This leads to greater flexibility. There is a clearly defined separation between components of a program—problems in each domain can be solved independently. New views and controllers (and hence devices) can be easily added without affecting the rest of the application.

Supporting Multi-channel access by MVC or n-tier architectures is considered to have a high impact on the software architecture.

When introducing Multi-channel Access in an application that does not already use layers or MVC (such as traditional monolithic applications), it means that a large part of existing functionality must be rewritten and reallocated. Responsibilities that are dispersed in a monolithic application need either be allocated to specific layers and to specific entities (client/server) or to specific controller and view components on specific devices.

7. Implementation

In order to provide Multi-channelling several mechanisms and techniques are required.

7.1. Being able to describe data independent from how it is displayed on a device

This is often done using XML (Extensible Markup Language). XML allows designers to create their own customized tags, enabling the definition,
transmission, validation, and interpretation of data between applications and
between organizations.

7.2. A mechanism to convert device independent data to device-specific
presentation

XSL Transformations (XSLT) is a language for transforming XML documents into
other another form. This could be another XML document, or a document in a
different format altogether, such as PDF, HTML, or even Braille. XSLT style
sheets work as a series of templates that produce the desired formatting effect
each time a given element is encountered. One of the most common uses of
XSLT is to apply presentational markup to a document based on rules relating to
the structural markup. Using XSLT XML objects can be transformed to a specific
format suitable for a specific channel (for example WML or HTML).

7.3. A mechanism to convert device independent data to device-specific
interaction and views

Generating the presentation for clients with different interaction models and flows
of control (for example WAP vs. Desktop) requires very different transformations.
Supporting such transformations increases development and runtime costs. A
more efficient solution is to define client specific controllers and views. This can be
done using Java Server Pages (JSP) and/or Struts. The Struts framework is a
flexible control layer based on standard technologies like Java Servlets,
JavaBeans, Resource Bundles, and XML, as well as various Jakarta Commons
packages. Struts encourages application architectures based on the Model 2
approach (also known as the JSP Model 2 architecture), a variation of the classic
MVC design paradigm. Several View and Controller components can be defined for different devices, while all target devices have a common Model component, which encapsulates business logic and data and is implemented as JavaBeans. To map this model on the persistent data some data abstraction layer is needed. Using Struts and JSP and/or XSLT the application can be converted to device-specific applications (viewsController) each of which contains one or more specialized JSP files. For example a dialog can be split into multiple specialized JSPs for devices with small screens. These specialized JSPs support the markup languages that their particular devices require, such as HTML or WML.

See [35,36] for more on implementing Multi-channel access with Java Server Pages and XML. Other types of implementations using MVC are also possible for example model 2X approach [37], NET MVC application framework [38,39].

7.4. A mechanism to assign devices to specific channels

Certain device profiles can be defined which describe individual devices detailed capabilities. A persistent storage device such as a database should record these definitions for different types of client devices such as mobile phones, PDA’s, and desktop clients.
# Index

---

**A**

ACD .......................................................... 31, 50  
ActionServlet ............................................ 33, 50  
AGD ......................................................... 31, 50  
Apache Struts Project........ See Struts Framework  
Application Customized Description........ See ACD  
Application Generic Description....... See AGD  
Architectural constraints..................... 17, 18  

**B**

Banavar et al ...............................................27  
BES45  
Blackberries.............................................. 9, 109  
Blackberry Enterprise Server ........ See BES  
Bridging patterns ................................. 12, 17, 36  

**C**

Cerqueria et al.............................................24  
Cocoon ........................................................... 32  
Composite design pattern .........................23  

**D**

design pattern ...........................................99  
Design pattern11, 12, 13, 14, 16, 17, 19, 20, 21,  
25, 26, 27, 30, 31, 35, 36, 40, 43, 48, 49, 50,  
51  
Device-independent ......................... 10, 29, 31, 49  

**E**

Evaluation of user interfaces .................16, 17  

---

**F**

Folmer .........................................................36  
Folmer et al............  12, 18, 19, 20, 36, 48  
Front Controller ........................................ 26  

**G**

GaiaOS .....................................................50, 51  
Gamma et al........................................ 18, 21, 22  
GNM9, 10, 11, 15, 19, 20, 36, 37, 38, 40, 43, 45,  
48, 51, 53  
Guardian News & Media Limited .......... See GNM  
GZIP ............................................. 10, 46, 47, 48  

**H**

Hess et al ..................................................29, 31  
Heterogeneity................. See Heterogeneous  
Heterogeneous10, 12, 14, 16, 21, 25, 29, 31, 33,  
35, 40, 43  
Heuristic evaluation .. 14, 17, 34, 38, 64, 65, 66, 67  
HTML ............................................... 26, 32, 48, 49, 51  
Human-Computer Interaction ............ 16  

**I**

ISO 13407 ..................................................33, 37  

**J**

Java Beans ..................................................32, 49, 51  
Java Server Pages ................... See JSP  
JDBC .......................................................32  
JSP .........................................................32, 51