

Adaptable User Interfaces for Diverse Human-Computer Interaction Devices

Wenjing Zou

Adaptable User Interfaces for Diverse Human-Computer Interaction Devices

Wenjing Zou

This thesis is submitted in fulfillment of the requirements for the degree of Master of Science in Computer Science at The University of Auckland.

January 2002

©2002 Wenjing Zou

Abstract

Many web-based information systems require degrees of adaptation of the system's user interfaces to different client devices, users and user tasks.

We describe a new approach to providing adaptable thin client interfaces for web-based information systems that allow a developer to specify a web-based interface using a high-level mark-up language. At run-time this single interface description is used to automatically provide an interface for multiple web devices e.g. desk-top HTML and mobile WML-based systems, as well as highlight, hide or disable interface elements depending on the current user or user task.

Our approach allows developers to much more easily construct and maintain web-based user interfaces than other current approaches. We describe an example application of our technique, the software architecture of our system, and its implementation using Java Server Page custom tag libraries. We report our experiences using the technology to build two web-based information systems.

We finally in this thesis present an evaluation of our Adaptable Approach and our comments on the potential future directions of this work.

Acknowledgements

First and foremost I want to thank for the guidance and support provided by my supervisor, John Grundy. He leads me to complete this thesis with such great idea, enthusiasm and excellent supervisor methods.

I would like to gratefully acknowledge the support of University of Auckland.

Thanks also to the nameless souls that spend time to complete my evaluation survey. Thanks their great comments they made on the questionnaire.

Heartfelt thanks go to my father who initially encourages me to complete my Master Degree in computer science. Also I would like to thank my boss on my work who always encourage and support me on my study.

Finally, special thanks to person who helped me with the final check before submission.

Table of Contents

Chapter 1	- Introduction.....	1
1.1	Introduction	1
1.2	Motivation	1
1.2.1	Fast Growing Use of Wireless Devices.....	1
1.2.2	Speed to Market.....	2
1.2.3	Reduced Number of Web Pages to Implement	2
1.3	Goals.....	3
1.4	Approach	4
1.5	Thesis Structure Overview	5
1.6	Summary.....	6
Chapter 2	- Related Work	7
2.1	Introduction	7
2.2	Developing Adaptable User Interfaces.....	7
2.2.1	User Interfaces Design for Various Computer Devices	7
2.2.2	Adaptable Systems	9
2.2.3	Developing Adaptable User Interface for Various Devices	10
2.3	Previous Work	11
2.3.1	Intelligent and Component-based Approach.....	11
2.3.2	Automated Converters.....	12
2.3.3	Synchronized Model-Based Design of Multiple User Interface	13
2.3.4	Palm's Web Clipping.....	13
2.3.5	XML/XSL Transformation.....	14
2.3.6	Open Custom Tags and Tag Libraries.....	15
2.3.7	Other of Previous Works	17
2.4	Summary.....	18
Chapter 3	- Thesis Related Technologies Overview.....	19
3.1	Introduction	19
3.2	Background.....	19
3.2.1	Overview of Diverse Human-Computer Device	19
3.2.1.1	Size of Screen and UI element	19
3.2.1.2	Input Methods.....	20
3.2.1.3	Memory	21
3.2.1.4	Speed	21
3.3	Overview of Web and Wireless Technology.....	22
3.3.1	WEB and WAP Architectures	22
3.3.2	WML and HTML	25
3.3.3	Develop WEB and WAP Applications.....	27
3.4	Summary.....	29

Chapter 4	- Traditional Web Applications For Various Devices.....	30
4.1	Introduction	30
4.2	User Requirement Specification.....	30
4.2.1	Requirement Specifications.....	30
4.2.1.1	Functional User Requirements	31
4.2.1.2	Non-Functional User Requirement.....	33
4.2.2	Use Case Diagrams.....	33
4.3	Object-Oriented Design.....	34
4.3.1	Software Architecture Design	35
4.3.2	Object-oriented Design.....	37
4.3.3	Interface Design.....	38
4.3.3.1	Design Interfaces for Different Devices.....	38
4.3.3.2	Design User and Task-based Interfaces.....	40
4.4	Implementation.....	42
4.4.1	JSP Basics.....	42
4.4.2	Database Implementation	43
4.4.3	Application Logic Implementation.....	44
4.4.3.1	Application Logic Flow.....	45
4.4.3.2	Get Device and User Information.....	47
4.4.4	User Interfaces Implementation.....	48
4.4.4.1	Developing Interfaces for WEB and WAP Enabled Devices.....	49
4.4.4.2	Examples of User Interfaces.....	50
4.4.4.3	User and Task-based Content Display.....	53
4.5	Running and Deploying the Application.....	56
4.6	Summary.....	57
Chapter 5	- Adaptable User Interface Technology	58
5.1	Introduction	58
5.2	Motivation for Developing an Adaptable Approach.....	58
5.2.1	Need to Develop Large Number of Interfaces.....	58
5.2.2	Hard coded for Each JSP	59
5.2.3	Fast Growing Devices	59
5.2.4	Bad Code Reusable Ability	59
5.2.5	Lack of Consistency	60
5.3	System Requirements and Specification	60
5.3.1	Functional Requirements.....	60
5.3.1.1	Write Once, Run Everywhere.....	60
5.3.1.2	Adaptation to User and User preferences.....	61
5.3.1.3	Adaptation to Tasks that Users Perform.....	62
5.3.1.4	Adaptation to Client Devices.....	62
5.3.1.5	Adaptation on Existing Client-Server Architecture.....	62
5.3.1.6	Configuration Capability	63
5.3.2	Non-functional Requirements.....	63
5.3.2.1	Easy to Use	63

5.3.2.2	Easy Read and Maintain	64
5.3.2.3	Performance	64
5.3.2.4	Programming Productivity	65
5.3.2.5	Extensibility	65
5.4	System Analysis	65
5.4.1	Adaptation Ability to Various Markup Languages	65
5.4.2	Screen Size and Page Content	66
5.4.3	Cookie Problems.....	67
5.5	Design and Algorithm	67
5.5.1	System Architecture Design	68
5.5.2	Designing AUIT Elements	69
5.5.2.1	Document Structure Elements	69
5.5.2.2	Generic UI Elements	70
5.5.2.3	UI Control Elements.....	72
5.5.3	Adaptation to Various Screen Sizes	73
5.5.3.1	Approaches	73
5.5.3.2	Design Elements to Perform Screen “Splitting”.....	75
5.5.3.3	Algorithm.....	78
5.6	Summary.....	79
Chapter 6 - Implementation of The AUIT		80
6.1	Introduction	80
6.2	JSP Custom Tags	80
6.2.1	Overview of Tags	80
6.2.2	Benefits of JSP Custom Tags	81
6.2.3	Defining the Tags	82
6.2.4	Build and Describe Tags.....	82
6.2.5	How our JSP custom tag-implemented AUIT page work.....	85
6.3	Implementation of AUIT tags.....	85
6.3.1	Build Document Structure Tags -Template Tag.....	87
6.3.1.1	Identifying the Client’s Capabilities to Serve Appropriate Content.....	87
6.3.1.2	Specify AUIT Document Structure	88
6.3.1.3	Output the layout	89
6.3.1.4	Sequence Diagram and Description	90
6.3.2	Build Page Flow Control Elements	90
6.3.2.1	Using “Group/Grouptr/Grouptrd” to Control Layout.....	91
6.3.2.2	Showing the Table in a Card	95
6.3.3	Build UI Control Elements.....	96
6.3.3.1	Build User/Task/Device Content Control Tags.....	96
6.3.3.2	Table Elements	100
6.3.3.3	Building Other Functional Tags –“Iteration”	101
6.3.4	Building UI Elements	102
6.3.4.1	Text and Layout Formatting Elements	102
6.3.4.2	Building Form and User Input Elements.....	104

6.3.4.3	Build Navigation Elements.....	106
6.3.5	System Deployment.....	107
6.4	Building Interfaces to Make the System Configurable	107
6.4.1	Introduction	107
6.4.2	Preference-based interface configuration	108
6.4.3	Examples	109
6.4.4	How to use these interfaces	111
6.5	Summary.....	111
Chapter 7	- Case Study	112
7.1	Introduction	112
7.2	Requirement Specification	112
7.2.1	User Requirements	112
7.2.2	Use Case Diagram	113
7.3	Case Analysis and Design	114
7.3.1	Object-Oriented Analysis (OOA).....	114
7.3.2	Software Architecture Design	115
7.3.3	Object-Oriented Design (OOD).....	116
7.3.4	User Interface Design	116
7.4	Implementation.....	119
7.4.1	Database Design	119
7.4.2	Implementation Steps	120
7.4.3	System Environment.....	120
7.4.4	Implementing the Applications Using the AUIT Tags.....	121
7.4.4.1	Implementing “JoblistInterface”.....	122
7.4.4.2	Implementing “JobdetailInterface”.....	124
7.4.4.3	Implementing Other Pages	127
7.5	Summary.....	128
Chapter 8	Evaluation	129
8.1	Introduction	129
8.2	Evaluation Topic.....	129
8.2.1	Evaluation from developers’ perspective	130
8.2.1.1	The Usefulness of Adaptable Approach.....	130
8.2.1.2	About Implementation of the Adaptable Approach	132
8.2.1.3	Ease of use of AUIT tags.....	132
8.2.1.4	Configuration Ability Evaluation	134
8.2.1.5	Programming Productivity	134
8.2.1.6	Code Maintenance Issues	134
8.2.2	Evaluation from End User Point of View.....	135
8.2.2.1	Ease of Use of User Interfaces	135
8.2.2.2	Acceptable Navigation Approach.....	135
8.2.2.3	Screen Response	135
8.2.2.4	Graphic Layout.....	135
8.3	Survey.....	136

8.4	Evaluation Conclusion.....	137
8.4.1	Evaluation Conclusion From the Developers.....	137
8.4.1.1	Comment about the AUIT tags.....	137
8.4.1.2	Functionalities of the AUIT tags.....	140
8.4.1.3	Comment About Programming Productivities.....	140
8.4.1.4	Comment about Code Maintenances.....	141
8.4.2	Evaluation Conclusion from End-user Perspective.....	141
8.4.2.1	Comment about AUIT Interfaces.....	142
8.4.2.2	Comment about the Display Methods on the AUIT Interfaces.....	142
8.5	Summary.....	143
Chapter 9	- Conclusion and Future Work.....	144
9.1	Introduction.....	144
9.2	Contributions of the Thesis.....	144
9.2.1	Easy to Use.....	144
9.2.2	Creating High-level Languages.....	144
9.2.3	Powerful Adaptability.....	145
9.2.4	Easy Architecture.....	145
9.2.5	Extensibility.....	145
9.2.6	Good Productivity.....	145
9.3	Future Work.....	145
9.3.1	Developing a Tool to Create AUIT Pages.....	146
9.3.2	AUIT Extensions.....	147
9.3.2.1	Add More Functions for the AUIT Tags.....	147
9.3.2.2	Add More Tags.....	148
9.3.2.3	Alternative Implementation Approach.....	148
9.4	General Summary.....	148
Appendix A	Bibliography.....	A-1
Appendix B	Evaluation Tutorial and Survey.....	B-1
1.	Introduction.....	B-1
2.	Background.....	B-1
3.	Evaluation for Group One – Developers Perspective.....	B-2
3.1	How to Run the Application.....	B-2
3.2	Tutorial.....	B-4
3.2	Survey Questions.....	B-8
4.	Evaluation for Group Two.....	B-12
4.1	Tutorial.....	B-12
4.2	Survey Questions.....	B-14
Appendix C	– Tag Library Description.....	C-1

List of Figures

Figure 1.1	Example of our Goal for User Interface Adaptation.....	4
Figure 1.2	Thesis Structure.....	6
Figure 2.1	XML/XSL Transformations.....	15
Figure 2.2	Example of “WMLOn” and “WMLOff” Tags.....	16
Figure 2.3	Example of “WMLEscape” and “Escape” Tags.....	16
Figure 3.1	Different Screen Size for Different Device.....	20
Figure 3.2	Input Method of Various Devices.....	20
Figure 3.3	Memory Difference among Various devices.....	21
Figure 3.4	Speed Difference for Various Devices.....	22
Figure 3.5	“Thin” and “Thick” Client Architecture.....	23
Figure 3.6	The WAP Programming Model.....	23
Figure 3.7	Architectures of WEB and WAP.....	24
Figure 3.8	Basic Structures of WML and HTML Document.....	25
Figure 3.9	Language Comparisons of HTML and WML.....	27
Figure 3.10	Browsers for Various Devices.....	28
Figure 4.1	Use Case Diagram of Car Site.....	34
Figure 4.2	Software Architecture for the Car Site.....	36
Figure 4.3	OOD Diagram of Car Site.....	37
Figure 4.4	Presentation of Same WML code on Different Device.....	39
Figure 4.5	(a) and (b) Example of User-based Content Display.....	41
Figure 4.6	(a) and (b) Example of Task-based Content Display.....	42
Figure 4.7	Car Site ER Diagram.....	44
Figure 4.8	Part of Application Logic Flow.....	45
Figure 4.9	Page dump for Homepage for HTML and WML Enabled Device.....	46
Figure 4.10	Code Example and Page Dumps for Login page.....	50
Figure 4.11	Search Car Interface on Various Devices.....	51
Figure 4.12	Display Search Result Interface on Various Devices.....	52
Figure 4.13	Search Dealer Interface on Various Devices.....	52
Figure 4.14	Dealer List Interfaces on Various Devices.....	53
Figure 4.15	Example of Task-based Content Display.....	55
Figure 4.16	Code Example.....	55
Figure 4.17	Code Example.....	56
Figure 5.1	The Markup Language and Devices.....	66
Figure 5.2	General Architecture Using AUIT System.....	68
Figure 5.3	Document Structures of WML and HTML.....	70
Figure 5.4	Form and User Input Elements.....	71
Figure 5.5	Text and Text-format Elements.....	71
Figure 5.6	Navigation Element.....	71
Figure 5.7	UI control elements.....	73
Figure 5.8	Example about Screen Splitting.....	74

Figure 5.9	Example of Screen Splitting.....	75
Figure 5.10	Use of “group” Elements	76
Figure 5.11	Approach to Perform Screen “splitting”	77
Figure 6.1	Part of Code Example for a Simple Custom Tag Class and Explanation	83
Figure 6.2	Part of AUIT.tld and Description.....	84
Figure 6.3	How our JSP Custom tag-implemented AUIT Pages Work	85
Figure 6.4	Examples of AUIT tags and Corresponding HTML,WML Tags.	86
Figure 6.5	AUIT Page Structure.....	88
Figure 6.6	Part of Code in “templateTag” Class	88
Figure 6.7	Part of Code in “templateTag” Class	89
Figure 6.8	Sequence Diagram of “templateTag” Class.....	90
Figure 6.9	“Group/grouptr/grouptd” Tags and their Properties	91
Figure 6.10	Search Result Interface Displayed on a Standard Browser.....	92
Figure 6.11	Code Example for “Search Result” Page	93
Figure 6.12	Page Dumps for Screen Splitting Performed on Two Devices	93
Figure 6.13	Example to Show Links for Various Devices	94
Figure 6.14	Page Dump for Display Table in a Card	95
Figure 6.15	Code Example for “ user” Adaptation	97
Figure 6.16	“searchresult.jsp”	97
Figure 6.17	Code Example for Using <AUIT:task>	98
Figure 6.18	Page Dump and Code Example for “device” Adaptation	99
Figure 6.19	Explanation of the Task-based Adaptation	100
Figure 6.20	Page Dump and Code Example.....	101
Figure 6.21	Use <AUIT:device> to Control Text Display.....	103
Figure 6.22	Use “allowcut” to Control Text Display	103
Figure 6.23	Code Example using Form Elements.....	104
Figure 6.24	Code Example to Display “option/select” Elements.....	105
Figure 6.25	Code Example to Dpecify <AUIT:form>	106
Figure 6.26	Code Example of Navigation Elements	106
Figure 6.27	Code in “web.xml” File.....	107
Figure 6.27	E-R Diagram of System Required Data	109
Figure 6.28	Sample Interfaces for Input System Data.....	110
Figure 7.1	Example Use Cases of the Job Management System.....	113
Figure 7.2	Class Diagram of Job Management System.....	114
Figure 7.3.	Our Four-tier Web-based Information System Software Architecture	115
Figure 7.4	OOD Diagram	116
Figure 7.5	Examples of Adaptive Job Management System Screens.	118
Figure 7.6	User base Interface for Job Management System.....	118
Figure 7.7	E-R Diagram (1).....	119
Figure 7.8	E-R Diagram (2).....	120
Figure 7.9	System Deployments.....	121
Figure 7.10	Examples of Job Listing Screen Running on Multiple Devices.	122
Figure 7.11	Logical Structure of the AUIT Description for the “job listing”	123
Figure 7.12	Code Example	125

Figure 7.13	Implement Job list Interface.....	126
Figure 7.14	Examples of Adapted Job Details and its AUIT Description.	127
Figure 7.15	Task-based Interfaces.....	128
Figure 8.1	Page Shots Using Adaptable Approach and Non-adaptable Approach	136
Figure 8.2	Evaluator and Tasks	136
Figure 8.3	Comparison of Usage between HTML and AUIT Tags	138
Figure 8.4	Bar Chart Diagram for Evaluation Result of UI Control Elements	138
Figure 8.5	Comments on the UI Control Elements from Evaluators	139
Figure 8.6	Bar Chart of Evaluation about “Group/Grouptr/Grouptd” Elements.....	139
Figure 8.7	Evaluation Result from End-User Perspective	142
Figure B.1	System Deployment.....	B-3
Figure B.2	“Template” Tag Description.....	B-4
Figure B.3	Form and User Input Elements.....	B-5
Figure B.4	Text and Text Format Elements	B-5
Figure B.5	UI Control Elements.....	B-6
Figure B.6	Page Shot for “Search_car”	B-7
Figure B.7	Page Shot for “Search_result”	B-8
Figure B.8	Page Shots Using Traditional Approach	B-12
Figure B.9	Page Shots Using Adaptable Approach.....	B-13
Figure B.10	Page Dump Using Two Approaches	B-14

Chapter 1 - Introduction

1.1 Introduction

Many web-based information systems require some degree of adaptation of the system's user interfaces to different client devices, users and user tasks [Van der Donckt et al 2001;Petrovski and Grundy, 2001]. Adaptable User Interfaces Development for PC was a popular research topic and has achieved quite good results. However with the fast growth of human computer devices, this topic now requires expansion to accommodate current and future requirements.

In this thesis, we aim to explore the development of adaptable user interfaces that allow software developers to more easily design, build and deploy interfaces for a wide range of users and user devices. We will focus on dynamic adaptation of User Interfaces for PC and other human computer devices like PDAs and mobile phones.

Our approach for user interface adaptation is based on the specific device features and includes the adaptation of the presentation format (e.g. HTML, WML driven browsers) as well as the contents, the user and the particular task.

In the rest of this chapter, we will introduce the motivation and goal of this thesis and briefly present the structure on how this thesis is organized.

1.2 Motivation

The main purpose of our research is to aid UI designers who are developing web-based applications for various devices. The following presents several main motivations for this research.

1.2.1 Fast Growing Use of Wireless Devices

One of the most important technological evolutions at the beginning of the 21st century is to combine the mobility offered by human computer devices such as mobile phones and

the enormous amount of information available on the Internet [Nokia Developer]. There is an increasingly varied range of devices through which humans interact with computers. These include desktop applications, web-based interfaces, mobile phones, Personal Digital Assistants (PDAs), pagers and so on.

More and more people are buying wireless handheld devices such as cellular phones and personal digital assistants (PDAs) and want access to online resources at anytime and anywhere.

1.2.2 Speed to Market

Each project has certain timelines, especially in today's fast growing e-commerce climate. The faster we make use of our information technology, the greater the chance of our business being successful. So the development time is a key issue for most businesses.

A key advantage of the adaptable approach is that developers can reduce development time by a "write-once, run-anywhere" interface approach.

1.2.3 Reduced Number of Web Pages to Implement

If a developer requires a web page currently designed for PC to be used by a wireless device such as a mobile phone, it is likely that he/she will need to redesign and rewrite the interfaces completely. This is not only because the programming language is different, e.g. mobile phones support WML, but also because other constraints, such as computing platform and network, device screen size, have also changed. There is also the need for the user interface to adapt to different users and user tasks, for example, hiding an "Update" button from some users such as customers, whilst allowing others, such as staff or administrator to view and access it.

This means that there is need to write pages for each different combination of user, user task and device based on device window size or languages that it supports. This approach has been taken surprisingly often. However, when the content of one's service is changing

frequently, keeping all services up to date and consistent is a demanding and time-consuming task.

This results in requiring a large number of interfaces to be developed and then maintained. Moreover, writing specific page code for each device makes a web site difficult to maintain, with the attendant lack of consistency. The back end access calls are repeated in each page code.

Therefore, this is a new challenge for user interface design and development: user interfaces need to accommodate the capabilities of various access devices and be suitable for different contexts of use, while preserving consistency and usability.

1.3 Goals

We aim to create a new approach that allows a developer to specify a web-based interface using a high-level mark-up language to develop adaptable thin-client interfaces for web-based information systems.

At run-time this single interface description is used to automatically provide an interface for multiple web devices e.g. desktop HTML and mobile WML-based systems, as well as highlight, hide or disable interface elements depending on the current user or user task.

Our basic goals are to support user interface adaptation across different users, user tasks, display devices, and networks (local area, high reliability and bandwidth vs. wide-area, low bandwidth and reliability [Rodden et al, 1998]) means a unified approach to supporting such adaptability is desired by developers [Van der Donckt et al 2001; Petrovski and Grundy, 2001].

We intend to develop a UI Adaptable System to fulfill the task of providing services for different devices with various screen sizes. The system should automatically adapt to the user's device and the user's preferences and perform the necessary scaling and conversion to the required contents format (HTML, WML, etc). [Korva 00]

We will use the following example of a “cars-for-sale” website to explain more clearly about our goal. In this site, if we write a page called “Search Result Page”, we want this page to perform several adaptations shown in Figure 1.1. If a user uses a non-PC device with small window size to browse this page, we require the page to be shown in a format which is legible within the screen limitations and consistent from a process and navigation perspective with what the user would expect to see on a PC. The result is shown on screen (1). If a user is a registered dealer, he/she should be able to see screen (2) with the “Update” link to update this car. If different task like “search a dealer” is performed on this “Search Result Page”, the screen (3) is shown.

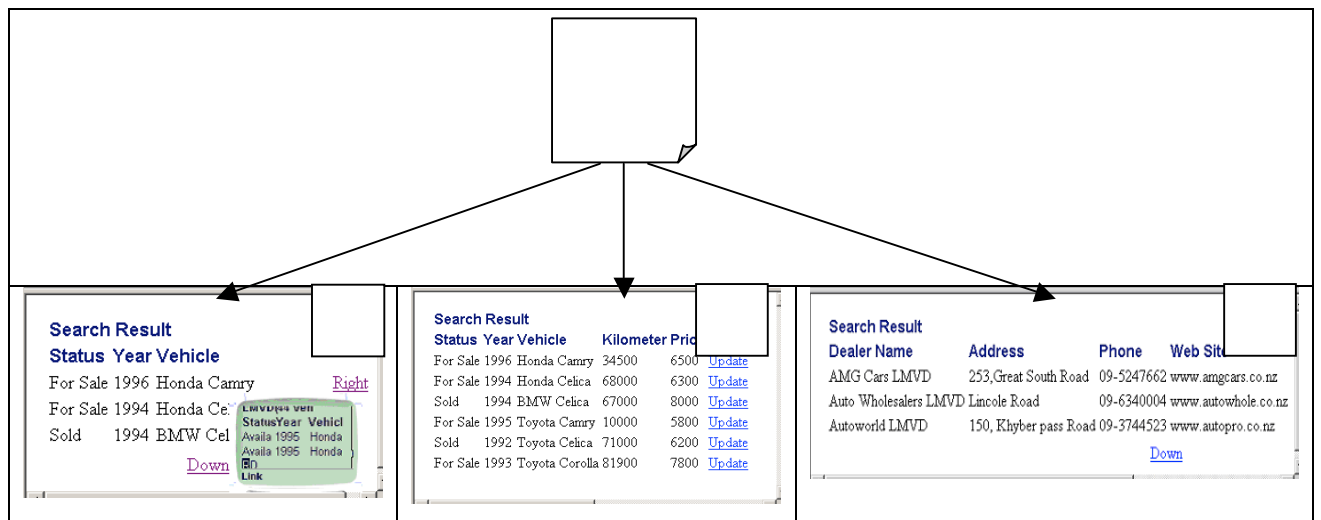


Figure 1.1 Example of our goal for user interface adaptation

1.4 Approach

We started with the researches on the current approaches and technologies for developing web-based applications for various human devices. We have summarised previous works and studied the relevant concepts and technologies. For practice, we have also gained experience from an example to develop an application using the traditional approach for various human computer devices.

From reviewing the current literature on these works, we formed our own ideas about the architecture of an adaptable approach.

Based on our own ideas, and on those from related work done by others, we have designed a set of device-independent mark-up language elements describing screen and layout, along with any required dynamic content (such as Java code) to specify User Interfaces. Screen element descriptions may include annotations indicating which user(s) and user task(s) the elements are relevant to.

In this thesis, we mainly focus on the considerations involved in the development of the adaptation methodology. The architecture of the system is also briefly described.

With adaptable approaches, a compromise is sometimes needed between the need to meet a limited development deadline and spending a huge amount of time developing a pretty user interface.

We then found a technology that is JSP custom tags to implement the creation of a set of such elements. We have also developed two Information Systems using those elements, namely a car site and a job management system.

The final stage of our work for this thesis is the evaluation of our approach and to get feedback from the users. We have also discussed how people can extend and improve our work in future.

1.5 Thesis Structure Overview

This thesis document is organised as follows:

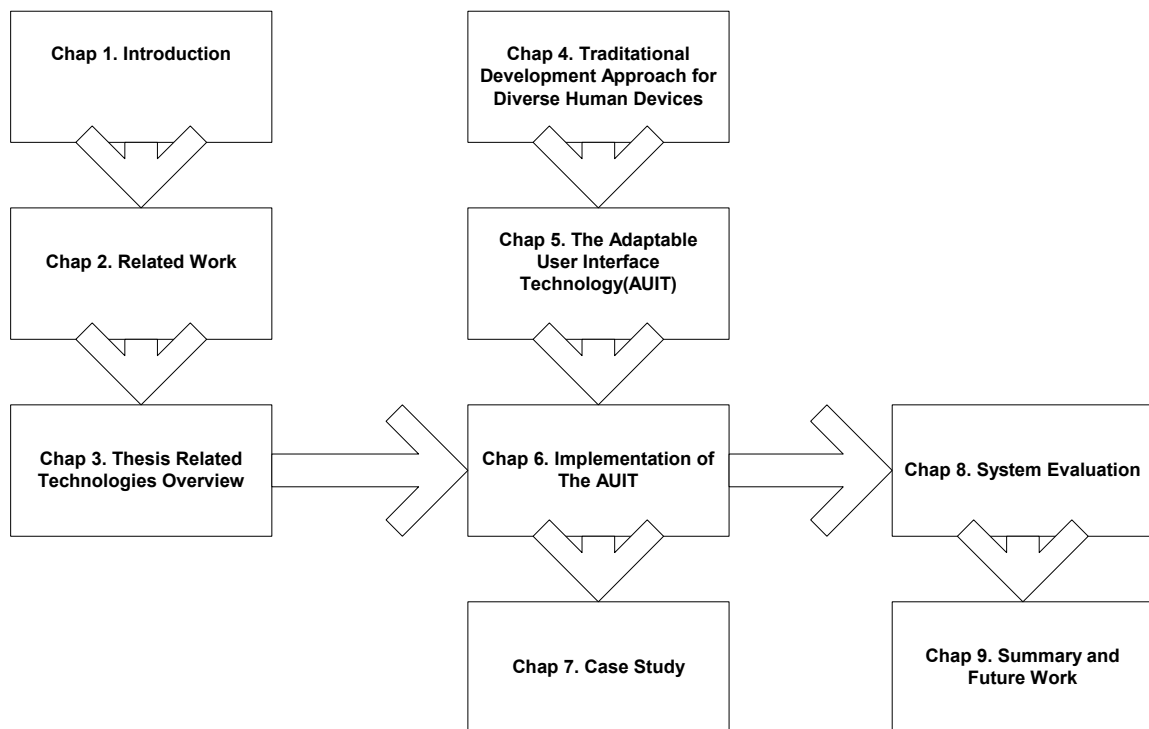


Figure 1.2 Thesis Structure

1.6 Summary

In this chapter, we have documented our motivation and established the research objective of our thesis. We have also briefly introduced the approaches that we have taken to do this research. The thesis structure is also presented in diagrammatic form.

Chapter 2 - Related Work

2.1 Introduction

In this chapter, I will present some of the previous researches and work with similar goals to ours. We will start with the introduction of the basic concept about adaptable user interfaces for various devices, and then demonstrate some of the related works that other people have done.

2.2 Developing Adaptable User Interfaces

User interface design and development can be a very important and time-consuming part of the software development lifecycle. Therefore, many web-based information systems require degrees of adaptation of the system's user interfaces to different client devices, users and user tasks [Van der Donckt et al 2001;Petrovski and Grundy, 2001]. More and more research groups are focusing on approaches and tools to design adaptable user interfaces, demonstrating its importance.

2.2.1 User Interfaces Design for Various Computer Devices

For the purpose of developing an adaptable user interfaces, we would like to overview the user interface design for various human devices first. User interface design for desktops has been a critical issue for developers, designers and researchers around the world. The expansion of human computer devices presents a new task of the interface developers. Functions that are usually performed easily on a desktop Web browser become awkward on a smaller device.

The development of the graphical user interface has made computers accessible to a wide range of users, but good user interfaces are still difficult to develop and there are still many challenges to be met before all the requirements can be satisfied. No single interface will satisfy every user. Users always have different needs as they learn to use an interface.

There are a many rules to follow when designing user interfaces for a system. The following lists three of them. In the real world, interface design can be very complicated task, depending a lot of factors.

- **Consistency**

Whenever designing a user interface, make sure layout and interaction are CONSISTENT across parts of an interface and between interfaces. This is an important aspect among the issues for designing of user interfaces. It's especially helpful if the user interface of an application is consistent with other applications on the devices, so users can work with familiar patterns.

- **Simplicity**

The layout of application screens needs to be simple so that the user can pick up the product and use it effectively after a short time. If they become complex or have too many functions, we can split them up.

- **Intuitive**

Make the interface behave, as the user would expect it to. This includes using metaphors appropriate to the problem domain and dialogue that users of the software use. [Tutorial SA and OOD]

There is no exception when developing user interface for devices with small window size. Mobile devices demand Web site content that is tailored to provide easy interaction while a person is on the move. Developers enabling a Web site for wireless devices need to consider in their design that some content is not suitable for wireless devices; for example rich multimedia presentations and large documents.

The design of interfaces for small screen devices requires special considerations due to a few distinct limitations.

- Firstly, handheld devices have a smaller screen size and resolution. A cell phone may only have a few lines of textual display, with each line containing eight to 12 characters.
- Secondly, cellular phones have a numeric keypad and several additional function keys. A more sophisticated device may have software-programmable buttons, but there are no keyboards or mouse in the wireless world.
- Third, handheld devices and cellular phones have limited computational resources. The low-power CPU and small memory size of these devices are often limited by power constraints.

2.2.2 Adaptable Systems

To achieve our goal of exploring adaptable approaches to develop user interfaces, we need to know about the concept of an “Adaptable System”. Adaptation is the process of responding to physical and mental characteristics of the user and his/her terminal equipment [Korva 00]. The basic principle of an adaptable user interface is universality of access irrespective of the specific characteristics of devices (e.g. display size).

Some user interfaces adaptations may require that user interfaces accommodate users with a wide variety of security levels and types, or terminal equipment with different characteristics. The concept of this type of adaptation is that when the system detects the user's level, it can adjust the interface to optimize it for that user. For example, more functions could be activated or a more compact display could be enabled to show more information on the display.

Adaptable systems generally retain the full power of the system. [Kules 00], but hide some of the part of the system based on the certain kind of criteria. They sometimes cannot be fully visible at anytime if no criteria meet the entire requirement. In other word, few users are able to use the whole system and other groups are only able to see part of the system.

For example, within a job management system of an organization, user input permission varies by the position, and layout will be different for each user position. The manager might view all jobs among his/her department, but individual staff of this department may

only see his own jobs. As users move to other positions, they can have different permit of use of the application. Certain level users will typically view the portion of the system relevant to their job very well, but not others. So the system has to be flexible enough to accommodate changes in specific users access.

In traditional UI design, however, the designer needs to specify a single interface for each criteria at design time, whereas adaptive interfaces yields a set of models and rules for generating the interface at run time.

Ideally, an adaptable user interface system will provide interfaces that will run on conventional web browsers as well as wireless PDAs, mobile phones and pagers [Marsic, 2001a; Han et al 2000; Zarikas et al 2001], as well as adapting to different user and user tasks [Eisenstein and Puerta, 2000], for example hiding an “Update” button if the user is a customer or a staff member doing an information retrieval task. However, building such interfaces using current web-based systems implementation technologies is difficult, time-consuming and results in hard-to-maintain solutions.

2.2.3 Developing Adaptable User Interface for Various Devices

From the goal of our system, we can see that there are several aspects of adaptation are involved in developing adaptable user interface for various devices. For various reasons, we want to be able to adapt a single document to suit different conditions and user need:

- **Target Platform:** a document may need to be adapted to the characteristics of an execution platform: screen size, supporting format, processing power. [Frank Rousseau] For example, some devices support HTML, and the others like mobile phone support WML.
- **Accessibility constraints:** a document may need to be adapted to different accessibility constraints: a browser may present alternate content or disable part of content based on a user level, task performed and etc.
- **User Needs:** the user may want personalized version of a document, so that we need to customize the document depending on various constraints.

2.3 Previous Work

After we have introduced the basic concepts of the “user interface design”, “adaptable system” and “developing adaptable user interfaces”, we will introduce some of the approaches and researches that people have been studying, and how they have achieved similar results as us. These researchers might have only achieved one aspect of our objective areas. For example, much work has been done in the domain of format adaptation: how to adapt encoding of a given media type.

2.3.1 Intelligent and Component-based Approach

In recent years, there has been an increasing interest in the use of component-based software architectures. These architectures use the notion of a software component object, which publishes its methods, properties and events for use by other components, and use large-scale component composition to build up software applications. [Grundy et al]

Various software architectures have been developed using components, including JavaBeans, COM/DCOM and OpenDoc. Tools allowing such architectures to be used to specify components and component-based applications include Jbuilder, Visual Javascript and Visual Age.

Intelligent and component-based user interfaces often support adaptation to different users and/or user tasks [Stephanidis, 2001; Grundy and Hosking 2001]. Most existing approaches only provide thick-client interfaces (i.e. that run in the client device, not the server), and most provide no device adaptation capabilities.

These have given us some of the ideas on adapting user and task-based content for thin-client user interfaces, and moreover to provide user preferred interface. But these works all assumed the use of thick-client applications where client-side components perform adaptation to users and tasks. They did not take into account different display devices and networks.

2.3.2 Automated Converters

To achieve content transformation across various Markup languages that different devices support, a specialized gateway can be used to provide automatic translation of HTML content to WML content for WAP devices [Fox et al 1998, Palm, 2001].

Various simple automated converters are available in the today's software market to support generic transformation from HTML to WML. Web developers can make use of this software to build their web site and perform conversion automatically to make their applications support wireless devices.

The main advantages with this approach are:

- 1) Increasing productivities of developers, so as to improve speed to market.
- 2) Purchase cost is not too high.
- 3) This approach is independent from the original web site.

Some of automated converters perform conversion in a fully automated mode. They are called "non-Configurable automated Converters". "Phone.com" WAP Gateway is the one that includes an automated content converter, and "Argo ActiCate" can convert HTML to WML, and XML.

However they do not normally work well for some applications, mainly because of the hard-coded logic of conversions that do not provide optimum user interactivity with the web service in many situations. Moreover this approach suffers from the problem of many poor user interfaces being provided due to the fully automated nature of the gateway. These problems were very obvious on some of the "non-configurable automated converters".

Some of the configurable converters, such as "Oracle Portal-to Go", "Spyglass Prism" and etc, can overcome some of these disadvantages. They rely on developer input and some extra configuration and template design in order to customize the conversion result [Atlas Software Technologies].

Overall, these techniques have had only moderate success in trying to convert HTML to WML for WML browsers, but they do not support user, task adaptation.

2.3.3 Synchronized Model-Based Design of Multiple User Interface

Synchronized Model-Based Design of Multiple User Interfaces, in their work, propose a set of techniques that will aid UI designers to build UIs across several platforms, while respecting the unique constraints posed by each platform. [Jean Vanderdonckt]

Our main goal for developing adaptable user interfaces is to have one interface written, make it work for any size screen intelligently. With similar goals as us, they also attempt to intelligently transform a given user interface from one context to another one, thus providing support for the multiple user interfaces simultaneously.

To achieve their goal, they raised a concept called “presentation structure”. The designers need to predefine a set of “presentation structures” in a presentation model. The “presentation structures” can be automatically generated based on the constraints specified for a given device such as screen resolution, size and etc. Their solution to the problem is to create mappings between each platform and an appropriate presentation structure. They abstract several “presentation structures” for solving a specific domain of problem.

This research prompted the idea for adapting the content from large screen format to small screen format by splitting a page into several sub windows.

2.3.4 Palm’s Web Clipping

With Palm’s Web Clipping approach, the translation cuts much of the content of the HTML document out to produce a simplified WML version. Typically, Web Clippings are small, dynamically generated Web pages, created by a CGI script in response to a user query. [Eric Cook]

The goal of Web Clipping is to minimize both display requirements (to fit on the Palm’s screen), and bandwidth usage. Web clipping is different to the automated converter

introduced previously. It is not dealing with transferring Web pages, but rather “Web Clippings”.

Palm Computing uses a completely different model for user/server transactions. Developers need to write a Palm Query Application (PQA) that is basically a mini-Web site. It’s simply a document written in HTML, and then compiled by Palm’s Query Application Server. The HTML used to write either the PQA or the Web Clipping is technically a subset of HTML 3.2, with a few small additions and modifications. [Eric Cook]

This solution has several restrictions. For example, the user has to minimize the use of graphics, and keep PQA and Web Clippings as small as possible. Further it would not support a WML device.

2.3.5 XML/XSL Transformation

This is one option that we can choose to implement our adaptable approach. We can write one set of pages (JSP/Servlet) to generate XML formatted data instead of generating output for a specific device. Selecting a particular XSL style sheets depending on the device used for browsing. The systems take XML-described interface content and transform it into HTML or WML or other WML formats depending on the requesting device information. [Marsic, 2001a; Han et al 2000; Zarikas et al 2001]

XSLT is a transformational language standardized in W3C that can be used to transform XML data to HTML, PDF, or another XML format. For example, we can use XSLT to convert an XML document from a format used by one company to the format used by another company. To generate the HTML page using this approach, we need an XSL stylesheet and a method to apply the stylesheet.

Using this approach, “Altas Software Technologies, Inc.” has done some researches with similar goals to ours. Because the transformation by embedded code (such as JSP/Servlet) will always be done in the end, a framework can be developed in such a way that it hides the calls to the transformation engine from a JSP/Servlet as shown in Figure 2.1. The

JSP benefit of this is the independence of the JSP/Servlet code from a particular transformation engine -- that is, the code can easily be ported to another wireless framework or server (such as Oracle Portal to go). [Atlas Software Technologies]

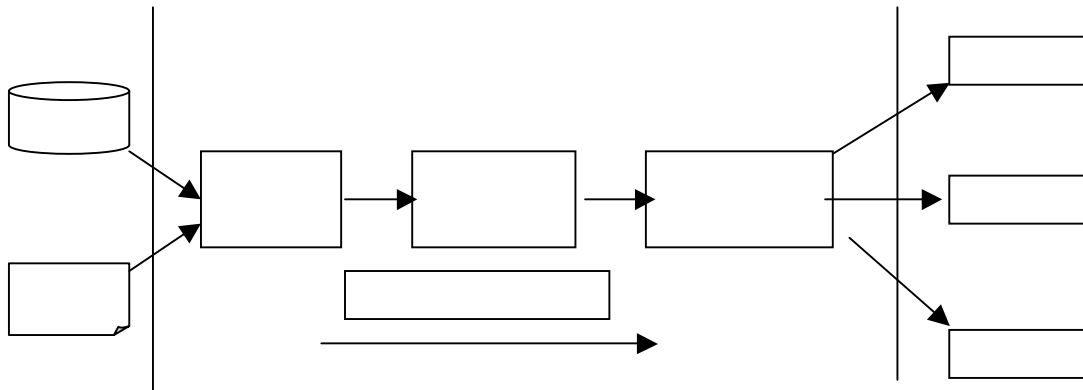


Figure 2.1 XML/XSL Transformations

This works reasonably well, but doesn't support user and task adaptation well and requires complex transformation scripts that have limited ability to produce good user interfaces across all possible rendering devices. The degree of adaptation supported is generally limited, however, and each interface type requires often complex, hard-to-maintain XSLT-based scripting.

Another downside of this approach is that we need to specify a stylesheet for each JSP, and based on that, perform the transformation. The stylesheet is a new language, with a complex syntax, and also has some limitations, and while it can fully present the layout format for each device, the developer needs to learn the new syntax.

2.3.6 Open Custom Tags and Tag Libraries

Developers can use JavaServer Page to develop web base applications. One great benefit of JSP is that it allows developers to create their own tags called "custom tag" which can provide more advanced forms of reuse of code by defining their own tag library.

On the following section, we will focus on presenting several such custom tag libraries to see how other people solve the problems as we faced

- WAP/WML Taglib

This is a JSP tags library that has been developed for WAP/WML developers. This library has two main tags “**WmlOn**” and “**WmlOff**” to lets people to combine within one JSP page different parts of code - for normal browser and for WAP browser. [jsptags[a)]

“WmlOn” tag	“WmlOff” tag
With <i>WmlOn</i> tag, we can mark part (parts) of our JSP code executed when our page is requested from WAP browser.	With <i>WmlOff</i> tag we can mark part (parts) of your code executed when your page is requested from normal browser.
<%@ taglib uri="taglib.tld" prefix="mobile" %>	
<pre><!-- HTML part --> <mobile:WmlOff> <html> <% out.println("
normal browser"); %> </html> </mobile:WmlOff></pre>	<pre><!-- WML part --> <mobile:WmlOn> <wml> <card id="test"> <p>You are using WAPbrowser</p> </card> </wml> </mobile:WmlOn></pre>

Figure 2.2 Example of “WMLOn” and “WMLOff” tags

But the functions provided by this library are quite simple and totally insufficient to meet our requirements.

- Codejava Escape Taglib

This is another JSP tags library that has similar goals as the previous one. It is also useful for WAP/WML developers to convert on the fly from HTML to WML. This tag library has got two body tags “**Escape**” and “**WmlEscape**”.

“Escape” tag	“WmlEscape” tag
<i>Escape</i> tag converts enclosing HTML tags to codes, so you can view them on WAP browsers. E.g.:	<i>WmlEscape</i> keeps HTML code as it is for non-WAP browser and converts it on the fly to the appropriate WML code otherwise. So it is a quick way to WML-ize your content. Just put this tag at the beginning of the each your page and all pages will be readable from the mobile phones! E.g.:
<pre><%@ taglib uri="taglib1.tld" prefix="esc" %> <esc:Escape> <html> your HTML code is here ... </html> </esc:Escape></pre>	<pre><%@ taglib uri="taglib1.tld" prefix="phone" %> <phone:WmlEscape> <html> Your HTML code is here ... </html> </phone:WmlEscape></pre>

Figure 2.3 Example of “WMLEscape” and “Escape” tags

Rather than just simply converting the HTML to WML when it is necessary, this library provides more powerful functions. Because WML files have got restricted size, this tag can split the content and generate appropriate JSP files “on the fly” (and link them together in WML cards). [jsptags(b)]

Developers need to set two parameters for **WmlEscape** tag: “directory” on their server for generated files (parameter name is *dir*) and appropriate “URL” for access to this directory (the parameter name is the *url*).

For example: let’s assume that the root directory of the web server that we use is “c:\inetpub”. You can create subdirectory “wmlpages” and use “WmlEscape” tag with parameters:

```
<%@ taglib uri="taglib1.tld" prefix="phone" %>
<phone:WmlEscape dir="c:\\inetpub\\wmlpages" url="http://your_host/wmlpages">
  <html>
    Your HTML code is here ...
  </html>
</phone:WmlEscape>
```

The adaptability that these two libraries can achieve is quite simple, but they give us indications that it is possible create an adaptable page with partial content specified by JSP custom tags to generate the proper Markup language for normal browsers and for WAP browsers.

2.3.7 Other of Previous Works

Other approaches include using a database of screen descriptions and to convert, at run-time, this information into a suitable mark-up for the rendering device, possibly including suitable adaptations for the user and their current task [Fox et al 1998; Zarikas et al, 2001]. This approach requires sophisticated tool support to populate the database and is quite different to most current server-side implementation technologies like JSPs, Servlets, ASPs and so on.

2.4 Summary

In this chapter, we have covered some of the background material and the previous works of other researchers.

Through overview of previous works we can see that most of the existing approaches have successfully achieved part of our goal. However they do have some limitations as follows:

- Some of existing approaches produce poor interfaces.
- The degree of adaptation supported is generally limited.
- Most existing approaches only provide thick-client interfaces (i.e. that run in the client device, not the server), and provide no device adaptation abilities.
- Some approaches require complex, hard-to-maintain XSLT-based scripting.

In our approach, we will avoid the problems and provide a high level and more powerful method of adaptation for user interfaces, which has more adaptation capability than any of the previous work. Our approach will make use some of the ideas from the previous works, such as using “tag” and user/task based thick client adaptation in conjunction with our own approach. Developers code an interface description using a set of generic, high-lever and device independent “tags”, which may also be annotated with information about the relevant user or user task.

Chapter 3 - Thesis Related Technologies Overview

3.1 Introduction

In chapter two, we introduced the prior related works. In this chapter, I will give a brief overview of the key technologies that are used in this thesis.

3.2 Background

The target terminal devices of our system are not just PC's, so we are going to overview the diverse Human-Computer devices and compare their main features with PC.

3.2.1 Overview of Diverse Human-Computer Device

Most Internet technologies have been designed for desktop and large computers running on reliable networks (with relatively high bandwidth). Today, access to the World Wide Web is not just limited to the desktop. Almost any handheld devices, including Psion, Palm Pilots, and mobile phones, provide users with a medium for accessing the Web services available today. [eMobile Part 2]

Excluding general desktop PC's, there are two main classes of wireless devices in existence in the market today:

- Pagers and Cell phones enabled for the Wireless Application Protocol (WAP)
- Palm devices, such as PDA, Palm Pilot and PocketPC

Among the devices that are widely in use by today's human being, we chose three main types of devices that are in use, (desktop, PDA and Mobile phone device) and will compare their features.

3.2.1.1 Size of Screen and UI element

Desktop	PDA (Palm OS)	Mobile Phone Device
15 to 17 inches Monitor, 1024x768 pixels, colour	Typical 160x160 pixels. PDA devices have a larger screen than a typical cell phone.	16 characters * 7 lines on today's typical cell phones. About 10 text lines per WAP "card". Monocolour

Figure 3.1 Different Screen Size for different device

This is one of the critical differences among various human interface devices. Apart from this, the size of each UI element rendered on these devices is also different. To develop our adaptable user interface we need to know relevant information about the size of the device and the size of each UI element on each device when developing the user interface.

For example, a standard character on desktop might take 5x5 pixels, and might take 7x7 pixels on PDA. So we can have a table in a database to store the information about the features of a particular device. This can help us to work out an algorithm regarding how much information to display on screen of a device. We will discuss the detailed algorithm in chapter five.

For this reason, developers need to design interfaces carefully with different priorities and goals from those used for large screens such as a desktop screen.

3.2.1.2 Input Methods

Desktop	PDA	Mobile phone Device
Keyboard and mouse	A pen to touch screen. User can either write Graffiti strokes or use the keyboard dialog provided on the device.	Limited input capability. There are number keys, which have alphanumeric coding & up and down keys.

Figure 3.2 Input Method of various devices

While Graffiti strokes and the keyboard dialog for Palm devices are useful ways of entering data, they are still not as convenient as using the full-sized desktop computer with its keyboard and mouse. The same applies to mobile phone devices.

With the different type of characters and input methods that each device has, the user interfaces must be designed differently based on these characteristics. For example, a PDA allows the use of a pen and touch screen, and desktops allow the use of a mouse, allowing more links on the interface to a PDA and desktop, but this will be problem of mobile phone device, because it uses key to direct link. Therefore, when we design the interface for these devices, we need to consider these aspects.

3.2.1.3 Memory

Desktop	PDA	Mobile Phone Device
Generally 64 MB + ram.	The Palm OS device has limited heap space and storage space. Different versions of the device have between 512k and 8MB total of dynamic memory and storage available. The device does not have a disk drive or PCMCIA Support.	Low bandwidth for data access. Less powerful CPU. Generally, 16 MB of RAM memory and 12 MB of ROM memory. For example, an Ericsson product called the Mobile Companion MC218, the CPU is a 32 bit ARM710T core which works at 36.864 MHz

Figure 3.3 Memory Difference among various devices

Because of the limited space and power, optimization is critical to make the application as fast and efficient as possible. When we design user interfaces, we need to consider the memory each device can support.

For example, on a “cars-for-sale” website, we allow a user to search cars based on “Make” criteria. On a desktop PC, we can provide a drop down list to display all “makes” that the user can choose. There will be more than one hundred car makes listed. This would be a problem for mobile device with low memory. We have to use “textfield” instead that allows the user to enter a “car make” they want to select.

3.2.1.4 Speed

Speed is a critical design objective for hand-held devices. The total time to navigate, select, and execute commands can have a big impact on overall efficiency. For

example, to maximize performance, the user interface can minimize navigation between windows, opening of dialog boxes, and so on.

Desktop	PDA	Mobile Phone Device
On a PC, users don't mind waiting a few seconds while an application loads because they plan to use the application for an extended amount of time.	The average Palm user uses a Palm application 15 to 20 times per day for much briefer periods of time, usually just a few seconds. So it has to be fast to handle user requirements.	Like PDA, due to slow transmission speeds, typically 14.4 to 19.6 Kbps, we need to keep text and graphics down to a bare minimum to make mobile device fast as well.

Figure 3.4 Speed Difference for various devices

3.3 Overview of Web and Wireless Technology

From the comparison over the last section, we know that handheld devices tend to have less memory, less powerful CPUs, different input methods, and smaller displays. Moreover, wireless networks have less bandwidth and more latency compared to wired computer networks. In the following section, we will provide more information about wireless technologies.

3.3.1 WEB and WAP Architectures

We will start with the introduction of the basic client-server architecture. Client-server architecture is generally classified into “thick” and “thin” clients.

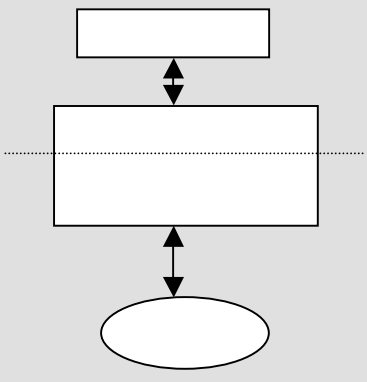
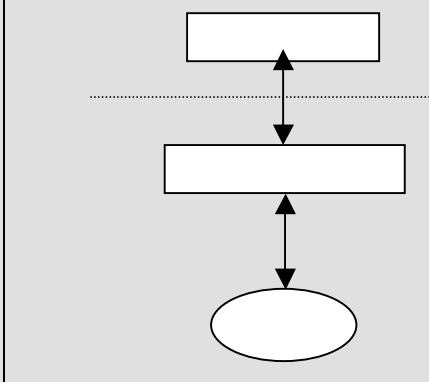
	Thick Client	Thin Client
Concept	Client-side does UI & some data processing. Server does some processing.	“Thin” clients are systems where client-side of the system is very limited e.g. Web browser rendering HTML, telnet session.
Benefit	Good for lower-volume networking, high-end clients.	Has the advantage of simple software installation and management on clients, ease of maintenance of the server-side programs, and simplicity. It suffers from limited user interface capabilities and performance.
Architecture		

Figure 3.5 “Thin” and “Thick” Client Architecture

In our research, we will mainly focus on the “thin” client architecture. To enable the web site to interact with wireless device, various gateways and protocols (for example, WAP and Palm VII Clipping proxies) are involved in wireless web access.

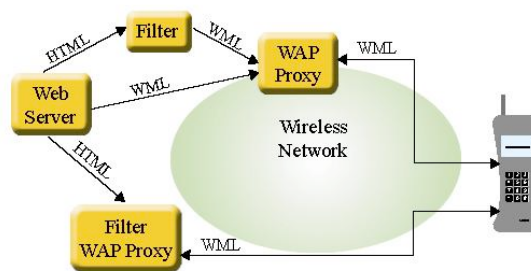


Figure 3.6 The WAP Programming Model

As you can see from Figure 3.6, the WAP programming model is based heavily on the Web programming model [Queay H.Mahmoud]. Some WAP gateways could be made to convert HTML pages into a format that can be displayed on wireless devices. But because HTML was not designed for small screens, the WAP protocol defines its own Markup language (WML). In most cases, the actual application or other content

located on the Web server will be native WAP created with WML or generated dynamically using some programming language such as Java servlets or JSP. In the following section, I will introduce the WAP in more detail.

	WEB	WAP
Model	<p>A client makes a request through its web browser to a certain web server corresponding to the URL précised in the request.</p> <p>In the Web server, the request is processed: Scripts are executed, HTML pages are fetched, etc, thus forming the content of the response to the request.</p> <p>This response is then sent back to the web browser of the client.</p>	<p>A client makes a request through the browser of its Wireless Application Environment (WAE), to a certain Web server.</p> <p>The request first travels in the wireless network to a gateway, which translates the request into HTTP and forwards it to the requested web server. In the web server, the treatment of the request is the same as in the World-Wide Web model. The content of the response is then sent back to the Gateway that translates it into WAP in order to send it back to the client.</p>
Application Layer	HTML, Scripting language	Wireless Application Environment
Transport Layer	HTTP,SSL,TCP,UDP	WSP,WTP,WTLS and WDP
Network Layer	IP	Bearer

Figure 3.7 Architectures of WEB and WAP

From the table shown in Figure 3.7, we can see the basic model difference between two main types of devices.

Wireless Application Protocol (WAP) is the gateway to a new world of mobile data. It provides a universal open standard for bringing Internet content and advanced services to mobile phones and other wireless devices [WAP Form Wireless]. It enables users to easily access Web-based interactive information services and applications from the screens of their mobile phones.

It has defined several standards such as the Wireless Markup Language (WML), which we will introduce in more detail in next section. WAP is a suite of specifications (including WML) that is based on variations of modern, open Web standards. WML, for example, is based on XML. [WAP Form Wireless]

WAP is the product of the WAP Forum (www.wapforum.org), an association founded in 1997 by Ericsson, Motorola, Nokia, and Phone.com (formerly Unwired Planet). [Giles Davies] The WAP Forum, now numbering more than 400 members, has contributed to the adoption of WAP and Wireless Markup Language (WML) as de facto standards. You will find WAP and WML in almost every Web-enabled digital mobile phone in the U.S. today, and all of the major communications service providers support WAP. You can still find remnants of older proprietary standards like Handheld Device Markup Language (HDML) supported in existing mobile phones, but the industry has settled on WAP for now [WAP Form Wireless].

3.3.2 WML and HTML

HTML has been well known by web developers and designers, but not WML. Wireless Markup Language (WML) is a tag-based display language providing navigational support, data input, hyperlinks, text and image presentation, and forms. WML is a browsing language similar to Internet html and became an “open standard” after WAP forum started. [Queay H.Mahmoud]

A valid WML deck is also a valid XML document, the same as HTML, and therefore we must design elements that contain an XML declaration and a document type declaration as follows.

WML	HTML
<code><?xml version="1.0"?></code>	<code><!-- 1 --></code>
<code><!DOCTYPE wml</code>	<code><html></code>
<code>PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"</code>	<code><head></code>
<code>"http://www.wapforum.org/DTD/wml_1.1.xml"></code>	...head information
<code><wml></code>	<code></head></code>
<code><card id="First_Card" title="First Card"></code>	<code><body></code>
<code><p></code>	...all other elements...
...all other elements...	<code></body></code>
<code></p></code>	<code></html></code>
<code></card></code>	
<code></wml></code>	

Figure 3.8 Basic Structures of WML and HTML Document

The following is a line-by-line explanation of the WML file example on the left side of table

- 1** The first line specifies the XML version number.
- 2** The second line specifies the SGML public document identifier.
- 3** The third line specifies the location of the WML document type definition (DTD). The DTD can be located in the network, or you can store it locally to make accessing it faster.
- 4** The fourth line defines the header of the WML deck. All WML decks must begin with a `<wml>` tag and end with a `</wml>` tag.
- 5** The following lines define a card containing a start and an end tag and text to be displayed to the user.
- 6** The last line is the deck footer. The user agent treats everything between the deck header and the deck footer as a single WML deck.

When a user agent loads the deck, it is displayed as shown below.

Apart from the basic structure difference between HTML and WML file, the table shown in Figure 3.9 provides some syntax comparison between HTML and WML as well.

From the comparison in Figure 3.9, we can see, WML supports text, images, user input, option lists, hyperlink navigation, and Unicode. WML is very like HTML. Many of the tags are the same, and familiarity with HTML, but with sufficient differences to be initially frustrating.

For example, in HTML, there are no functions to check the validity of user input or to generate messages and dialog boxes locally. To overcome this limitation, JavaScript was developed. Similarly, to overcome the same restrictions in WML, a new scripting language known as WMLScript has been developed.

	HTML	WML
Basic Unit	Page	Cards are grouped together into decks. A deck is the smallest unit of WML that a web server can send to a user agent.
Event	submit	Do,ontimer,onenterforward,onenterbackward,onpick,onevent
Graphic	 tag to support images. Image can be type of “.gif, .jpg” color image	 tag to support images. Support only “.wbmp” format, which stands for wireless bitmaps, and specifically designed for wireless device with a small monochrome screen.
Text Formatting	Font, and	Br,p,pre
Table	HTML table model allows to arrange data-text, preformatted text, image, links, forms, form fields, other tables, etc.	Table can contains tr,td tags. Only text and image are allowed in the cell.
User input	textarea, textfield, option, select and etc.	input, select, option, optgroup, fieldset
Tags	HTML permits the mixture of cases Less strictly enforced in HTML	All tags must be in lower case. All tags must be completed.

Figure 3.9 Language Comparisons of HTML and WML

3.3.3 Develop WEB and WAP Applications

Most of the devices provide ability to access the information through Internet. Instead of using Wireless Markup Language (WML), some of devices use a subset of HTML as the content language, so that it could not fully accept HTML functions, Eg. rich multimedia presentations and large documents. For example, PalmVII with built-in wireless modem uses Palm Query Application (PQA, a subset of HTML). PocketPC uses Microsoft Internet Explorer (PocketPC version) for browsing HTML content.

As most web developers know, web based applications can be implemented by a wide range of technologies. Each of them is produced by a vendor, each has a certain

amount of market share and each has their own advantages and disadvantages. With WAP applications, which can also be hosted on normal Web servers, is less known than web applications. Therefore not many people are familiar with WML application development.

WAP applications can be written in WML and WMLScript. We can also write them using existing Web technologies. CGI scripts, servlets, JavaServer Pages, Active Server Pages, Perl, Tcl, and so forth all can generate dynamic WML documents. [Queay H.Mahmoud]. In other words, all of these technologies can be used to develop WAP application.

For example, Java Server Pages allows us to embed Java statements within HTML documents. When JSP is invoked, it is compiled into a Java servlet and executed by the server to create a dynamic HTML document [Giles Davies]. In the case of WAP, it just simply creates a WML document instead. Therefore, developing the WAP applications using JSP can be done easily once we know the syntax of. In the next chapter, I am going to demonstrate how WAP applications can be developed in JSP for a car site.

With a mobile device, user agents handle the interpreting of the content on behalf of the user. The WML browser is one such user agent. It is very similar to a web browser except that it handles content formatted in Wireless Markup Language (WML). User agents also typically have a built-in WMLScript Interpreter for running applications. A micro browser is usually used by WAP device to render the WML and /or WMLScript to the user.

Desktop	PDA	Mobile Device
Can use standard browser for desktop, such as IE and Netscape.	1) Can use browser such as AU-System WAP to access internet. 2) Instead of using WML, PDA also can support a subset of HTML as content languages	The Toolkit browser is used for the Nokia emulator. It can display URLs from WAP Gateway as well as local files.

Figure 3.10 Browsers for Various Devices

3.4 Summary

In this chapter, we have briefly introduced the features of mobile devices, WAP/WML, and the wireless technologies. Through the comparison of the respective features of devices, such as screens size, Markup Languages and browsers, we know the features of several devices and the wireless technologies that most people are not familiar with. From the issues introduced in this chapter we have built some basic knowledge in order to develop adaptable user interfaces.

Chapter 4 - Traditional Web Applications For Various Devices

4.1 Introduction

In this chapter, I will use a cars-for-sale site as an example to demonstrate how to use traditional technologies and approach to develop a web-based application and enable it for various human devices, such as desktop, Palm, mobile phone, etc. Some of the user and task based content display features and approaches are also demonstrated.

To develop this application, the following software development life cycle is involved.

1. Requirement Specifications.
2. Object-oriented Analysis (OOA).
3. Object-oriented Design (OOD).
4. Implementation.

The main purpose of this chapter is to give a motivating example for this thesis by reviewing current approaches. Solution to building adaptable, web-based information system user interfaces will be presented in following chapters.

4.2 User Requirement Specification

The Car Site is a commercial web site on which car dealers can advertise the cars they wish to sell. Web users can browse and search cars for purchase. The following is the detailed user requirement for this project.

4.2.1 Requirement Specifications

Determining the requirements for a project is usually the first step in the design phase within the software development lifecycle. This is also a very important aspect for system developers to consider.

4.2.1.1 Functional User Requirements

This section will address what the system should provide, such as interactivities and functionalities.

The main feature of the car site system is to provide user interfaces and functionalities based on the type of user, tasks the user performs and the devices the user needs to use. For instance, a dealer might wish to use a mobile phone to update their stock information. This will create a degree of complexity within the system development because different devices have different features and are good at certain functionalities.

In this system, I will assume that user might use three main types of devices to utilize this system; they are desktop PC, , Personal Digital Assistant (PDA) or mobile phone. From the chapter three, we have learned about the main features relating to each of those devices. This system has three main types of users, who can perform different tasks within the system. All users are able to use the same type of devices to view the system.

Actor One: General Web User

General web users are free to view the web site by entering the URL. Within the site they can access a variety of functions:

1) Search for cars based on preferences

Users can retrieve a list of cars that match the requirements specified by the user. The system can provide several levels of search.

- By Region: (Auckland, or a variety of other regions).

This is the first level of the search. The user can select any region the system provides, such as Auckland region, or Nationwide to conduct a search. This will narrow the number of results instead of give the whole list of cars within the database.

- By Make, Model, Year, Price of cars.

The system should provide a form with some fields for user to enter the criteria such as drop down list with car make or model information and car price range fields that a user wants to search. For example user can select “BMW” make from the “make” drop down list, and enter the price from \$10,000 to \$15,000. The search result will show a list of cars that match these criteria.

2) Search dealers.

User can retrieve a list of matching dealers. Web user can search all dealers within a particular region. The relevant dealer information, such as address, contact phones, etc, will be displayed on the site. Users can also search a particular dealer by entering a clue of a dealer via a keyword search field, such as dealer company name or address.

3) View all stock of particular dealer.

The user can view all the stock of a selected dealer.

4) Select a list of cars for comparison and buy a car

From a list of cars either from search results, or from a dealer’s stock list, the web users are able to select a shortlist of cars to view.

Actor Two: Car Dealers

Dealers can maintain their own stock listings, editing, updating and inserting & deleting cars.

Actor Three: Web Sales Staff

- 1) Update featured cars or banners displaying on home and other pages.
- 2) Load and Update dealers’ information, such as subscription information and so on.

4.2.1.2 Non-Functional User Requirement

1) User Interface requirement

Most of the web users will use desktop to view the web site, so that all the interfaces for those users need to be very attractive. Attractiveness of graphic layout is one of the important elements for successful web site. Because of the restriction of the screen size, the user interfaces displayed on the mobile device will not have to be very attractive.

2) Security requirement

These need to be considered in this system due to the different having varying levels of access to functions. For example, only the registered dealers can update own stock, but they can't have access to other people's listings or to the reporting systems. The sales reps of the site owner will be able to modify customer data and view the hits of the stock for the dealers they are in charge.

3) Easy to use, robustness.

The web sites should be easy to use, and search engine needs to be powerful and fast enough to deliver accurate results in reasonable time. The functionality of web site should suit any type of the end users, either dealers or general web users without the need for training. This is especially true regarding the user interfaces for mobile devices - they should provide logical, easy to use & follow functions for users.

4) Response time and Performance.

Whenever the user wants to search for cars by entering certain criteria, such as make and model, they expect the search results to display as soon as possible. This is most common requirement in most of the application development.

4.2.2 Use Case Diagrams

Use case diagrams are used to represent the interactions of the different entities in the application. It uses actors and use cases graphically to represent the main components in the application. The use case diagrams in Figure 4.1 show how three main actors who are general web users, dealers and web sales staff, could interact the system.

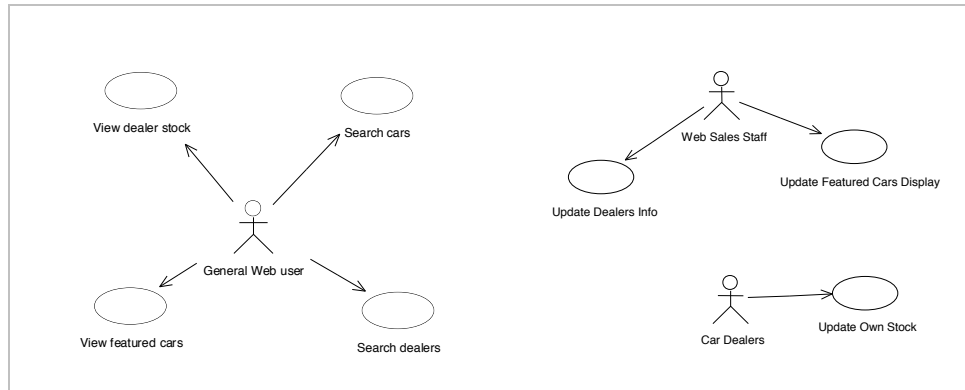


Figure 4.1 Use Case Diagram of Car Site

4.3 Object-Oriented Design

Design is an important phase within the whole stages of software development lifecycle. Design decisions will affect a lot of aspects such as how the system meets the requirements, how reusable the code is, performance, and so on. Bad design will cause the system several problems such as hard to maintenance difficulties as the size of the system grows, poor speed or performance, etc.

The main difference between the system we are going to develop with other normal web applications is that our client devices will be not just a general PC. We will enable this system to work with some wireless devices such as PDA's and mobile phones. These devices require a language other than HTML, and have special screen size constraints sizes and diverse display related requirements. Our design must consider these factors. Moreover, our system needs to consider the approaches to displaying task and user based content that will be explained in more detail in "section 4.4.3".

In this section, we will present our design proposal in three aspects: software architecture design, OO Design, page flow and interface design. The main feature of the design using a

“standard” approach is that we will need to have several versions of the interfaces developed for combinations of each device, end user type and tasks.

4.3.1 Software Architecture Design

Typically, when developers have a software development project in hand, a set of system architectures need to be determined. First of all, developers need to decide on the general architecture the application will adopt based on the features and requirements of the project.

In chapter three, we introduced the “thin” and “thick” general client and server architecture. From the requirement specification, we can see that our system will use the “thin” client architecture - that is, almost all data processing is on server side. The client side will be browsers.

This car site will adopt a 3-tiered architecture as shown in Figure 4.2 for flexibility and scalability reasons. The separation of the user interface, business logic, and data access allows clean code and easy maintenance capabilities at each tier without disturbing the others. The decoupling between the tiers allows us to add more capability to each tier at the system scales.

Another benefit of using this architecture to develop our car site is that we can easily add another presentation tier for the layout formatting (that will be introduced in the next chapter) without too much modification of system architecture.

Moreover, because most software developers are not good at the graphic design, web page development tools like “Dreamweaver” can assist the graphic designers to design graphic pages then pass the static pages to the web developer to fill in the code needed to make the page dynamic. With this architecture, web designers can work in parallel with the programmers, and make the development more efficient.

We chose JavaServer pages as the technology to develop this system. Therefore, in the architecture, we use “JavaBeans” to hold and process data. It will perform retrieving or

putting data into a data source object, and a number of JSPs to present the data. The JSPs are merely views of data exposed by middleware components. They will not directly access the system data objects.

The following diagram shows the architecture that we are going to use to develop this system.

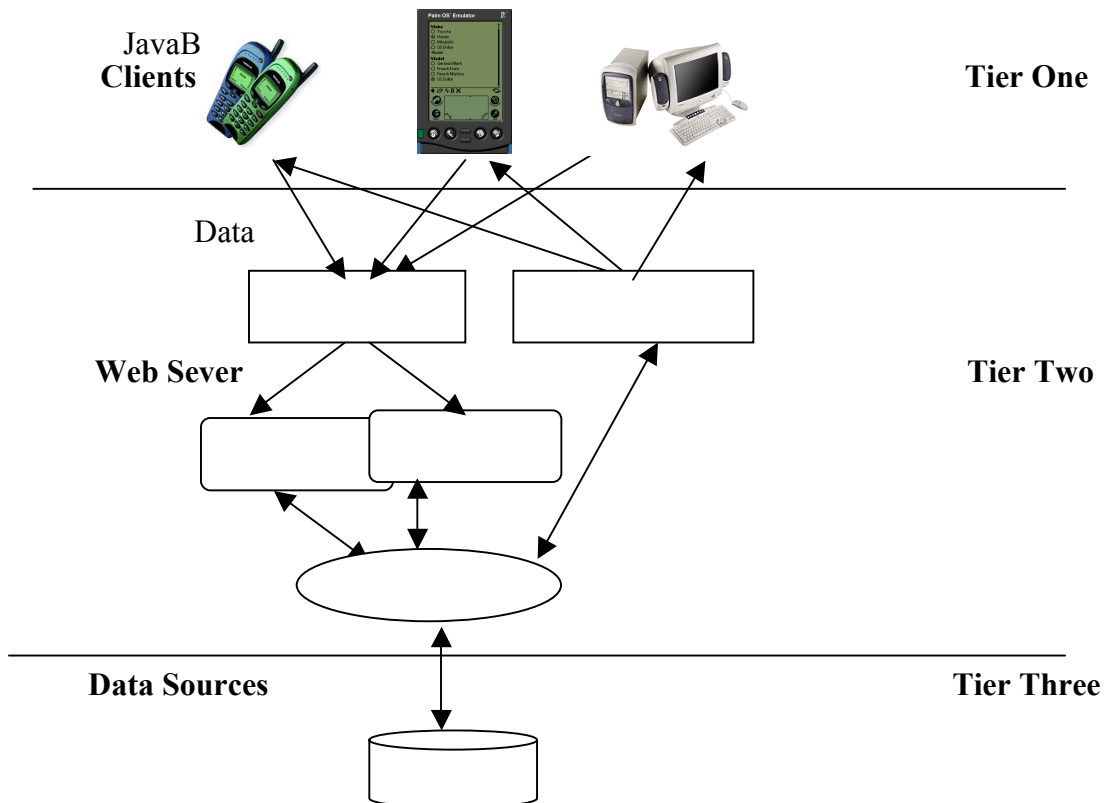


Figure 4.2 Software Architecture for the car site

This diagram in Figure 4.2 illustrates the application in three-tiers. Tier 1 is composed of multiple clients, which request services from the middle-tier server in tier 2. The middle tier server accesses data from the data source in tier 3, applies business rules to the data, and return the results to the clients in tier 1. When a user types a URL on the browser of a device, “index.jsp” is first retrieved, and functions will detect the type of a device that will retrieve the page. Thus, the relevant page will be sent to the browser. For example, if a mobile phone is retrieving a page, the JSPs that are developed for mobile phone device will be processed, and sent to the client.

The user interfaces handles user's interaction with the application; this can be a web browser such as IE or Netscape running through a firewall, a heavier desktop application or a wireless device, mobile phone or palm. The middle tier needs to support a variety of clients, such as Web browsers, and hand held devices. The client tier handles interfaces display, and do not query database, execute complex business rules.

4.3.2 Object-oriented Design

The next important step for design is to identify the application objects. The OOD diagram is a more detailed representation of the programs that are going to be used in the application. It shows the attributes and operations that each class provides, and relationships between classes. Let us to take a look how the OOD diagrams look with this thin, 3-tiered application.

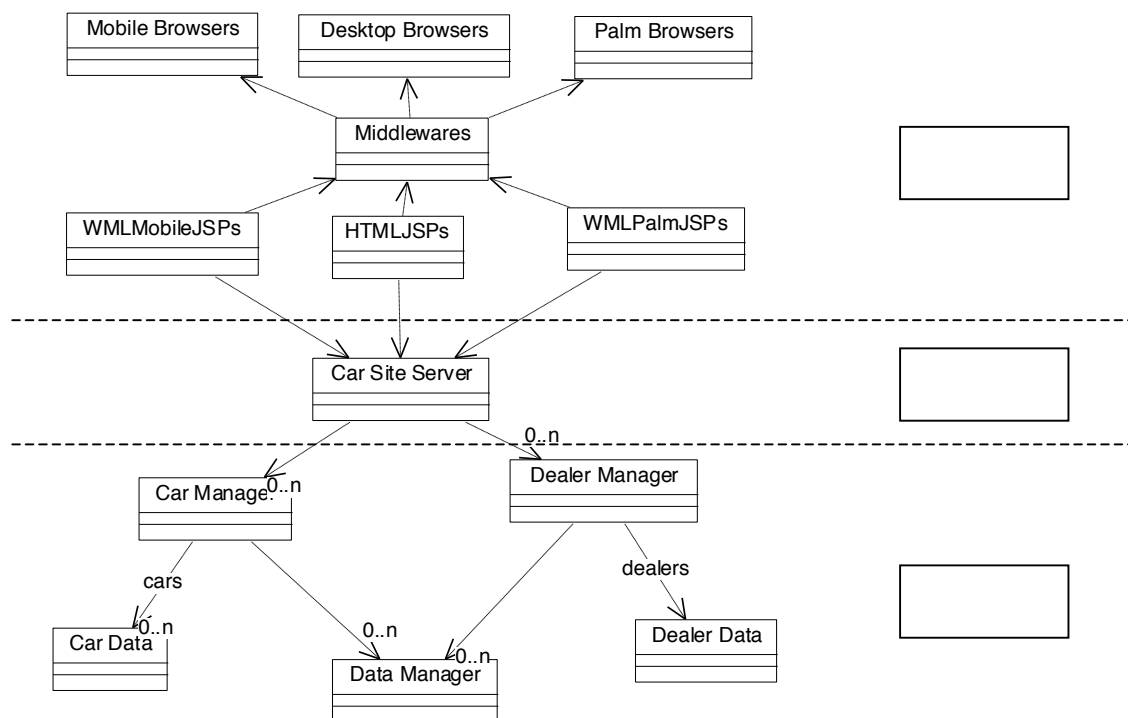


Figure 4.3 OOD Diagram of Car Site

A user goes to the JSP page, which runs on the server (Tier 2), by supplying the JSPs URL to the browsers on Tier 1. The JSP communicates directly with the database. This uses “Data Manager” objects that in turn access a relational database.

The “Data Manager” is a service object that basically corresponds to APIs to provide some basic functions to support the “Car Manager” and “Dealer Manager” objects. The “Car Data” and “Dealer Data” objects are collection classes for collecting records for car and dealer.

4.3.3 Interface Design

In this step of the design phase, the developer needs to work out the page flow using detailed screenshots or diagrammatical representations of each page based on the specified user requirement. Because the client devices are not just PCs, we need to design separate sets of interfaces for each device based on its specific features. Because we have more than one type of user and each of them can perform different tasks, we will give an example of how to build interfaces based on user and task.

The interface design for this application can be very time consuming task.

4.3.3.1 Design Interfaces for Different Devices

Designing web pages for desktop with standard window size using HTML can be easy. A lot of visual tools like “DreamWeaver” can also assist the interface developer to design the interface easily.

Small screen and pen-based user interaction require a different UI paradigm than a desktop computer. For example, the palm OS device’s screen is only 160x160, so the amount of information it can display at one time is limited [Palm OS]. Based on different screen sizes on different devices, developers need to design interfaces carefully with different priorities and goals than are used for large screens such as desktop.

The interface displayed on the mobile device or PDA can be completely different to an HTML page, even the font and layout of the components (such as buttons) may vary depending on the browser of the individual device. Developer need to have the knowledge of the markup languages such as WML that are used for rendering the interface of that device, and what the layout of each component will look like in a particular device. The

developer needs to design and manage the layout of the interface for those devices as well, but this can be very time-consuming job.

One reason is because we are currently short of tools to assist with design for each interface. Another reason is because the size of WML UI component displayed on the screen is different to the HTML UI component that we are familiar with. For example, the size of a button displayed using HTML on desktop browser can be very different with one using WML displayed on a PDA or a mobile phone.

Even though the developers need to spend so much time on developing each version of the interface, the advantage is that each interface can be designed differently therefore have more flexibility. For example, the same component in Markup language can have different behaviors (for example in IE and Netscape). That is why some of the sophisticated web sites have several different versions to handle the problem. The various interfaces will be shown based on the type of the browser. This is a known problem to most web developers, but fewer people know that the WML markup language has the same problems with the different types of devices.

For example, the Phone.com browser supports cookies whereas others don't. The Nokia WAP Toolkits browser will render the same WML with different layout as shown in Figure 4.4.



<pre> <select name="menu"> <option onpick="AcctList">Account List</option> <option onpick="TransferFundsHome">Transfer Funds</option> <option onpick="BillPaymentHome">Bill Payment</option> <option onpick="SignOff">Sign-Off</option> </select> </pre>	
	
<p>The code displayed on Phone.com WML browser.</p>	<p>This same code also works on other browsers (like Nokia), but in two stages.</p>

Figure 4.4 Presentation of same WML code on different device

What some of the currently successful web sites are doing to resolve this is writing three different versions of pages for these browsers. They use the same methods as we illustrated in previous sections to verify the browser header to get information about that browser so as to display the correct content. This is very time consuming, but can produce quite a good result.

Apart from the graphics using general html or other markup languages, we also need to have correct presentation of the back end data retrieved from database. This could be a problem for a wireless device with less memory. For example, we can have drop down list of car makes for user to choose their favorites cars, but can not use the same system have them for mobile phones, because the list could be a number of hundreds making the mobile short of memory. For this sample application, the full amount of data can be displayed on desktop using HTML with fully supported image display, but the same data cannot be fully displayed on the mobile device. An alternative solution needs to be found.

The web developers should have rule in mind that is to make embedded code as less as possible. That way, the resulting dynamic pages contain a minimum of logic, so that they will be intelligible to page designers required making any changes to the rendering of the data without making too much mess of the embedded code.

So that the developer needs to know the features that all the intended receiving devices have and design an appropriate set of interfaces to support each of them.

4.3.3.2 Design User and Task-based Interfaces

A user interface may show different layout or parts of interface to a user, due to the user's level of expertise, the task and/ or role being performed, the users' personal preferences. The concept of the user-based content display is using the same layout with some parts hidden based on the user. Now we will give some examples from our car site.

- **User based content display**

In the car site, we have the following situation: these two web pages discriminate content based on a user's role. The page (a) and (b) in Figure 4.5 are both showing a list of cars. The difference between these two pages is that page (a) has an extra column with an "Update" link on it. The page (a) in Figure 4.5 is only shown if the user is a registered "dealer" who will use this link to update the information about these cars for their stock. Page (b) will be shown when a general web user retrieves a list of cars that match the requirements specified by the user, such as color, make, model, and year.

Status	Year	Vehicle	Kms	Price	Update
For Sale	1994	Toyota Corolla LX Ltd	47561	8995	Update
For Sale	1992	Toyota Aristo	73417	17995	Update
For Sale	1995	Honda Rafaga	68830	11995	Update
For Sale	1993	Honda Prelude Si	86870	10995	Update
For Sale	1995	Toyota Corolla Ceres	53939	10995	Update
For Sale	1997	Nissan Wingroad JS 4WD Wagon	86865	13995	Update
For Sale	1994	Nissan Sunny Super Touring	76865	9995	Update
For Sale	1994	Subaru Legacy TX Wagon	108964	10995	Update
For Sale	1991	Nissan Pulsar X1R 5 Dr	49113	8995	Update
For Sale	1991	Nissan Pulsar X1R 3 Dr	92102	7995	Update
For Sale	1986	Mitsubishi Mirage Asti Coupe	106530	9995	Update

Status	Year	Vehicle	Kms	Price
For Sale	1994	Toyota Corolla LX Ltd	47561	8995
For Sale	1992	Toyota Aristo	73417	17995
For Sale	1995	Honda Rafaga	68830	11995
For Sale	1993	Honda Prelude Si	86870	10995
For Sale	1995	Toyota Corolla Ceres	53939	10995
For Sale	1997	Nissan Wingroad JS 4WD Wagon	86865	13995
For Sale	1994	Nissan Sunny Super Touring	76865	9995
For Sale	1994	Subaru Legacy TX Wagon	108964	10995
For Sale	1991	Nissan Pulsar X1R 5 Dr	49113	8995
For Sale	1991	Nissan Pulsar X1R 3 Dr	92102	7995
For Sale	1986	Mitsubishi Mirage Asti Coupe	106530	9995
For Sale	1994	Subaru Legacy Excimo	53454	11995
For Sale	1993	Subaru Legacy GT Wagon	89598	10995
For Sale	1995	Toyota Curren	77141	13995
For Sale	1991	Toyota Apex GT	75003	8995

Figure 4.5(a) and (b) Example of User based content display

Web applications often display content based on a user's role. Another example on our car site is an interface that displays a list of matching dealers retrieved by a user. General web users only see the general information, such as address and contact information. Web sales representatives can see additional information for each dealer such as the hit statistics and subscription information relating to that dealer. Dealers themselves have access to a button to update their own information such as address, phone number and so on.

- **Task-based content display**

We can also have the following situation, these two web pages discriminate content based on a task that a user can perform.

The figure shows two side-by-side screenshots of a web form titled "Dealer Information".

Form (a) - Update Dealer: This form is for updating an existing dealer. It contains the following fields:

- Company Name: AMG Cars
- Company Type: LMVD MVD BMVA Private
- Contact Name: Ion Sheriff
- Company Address: 253 Great South Road
- Suburb: Greenlane, City: Auckland
- Region: Auckland/Whangarei
- Postal Address: PO Box 17 158 Greenlane
- Email: sales@amgcars.co.nz
- Web Site: www.amgcars.co.nz
- Phone Number: 09 5247662
- Phone Number 2: 01
- Phone Number 3: 01
- Fax Number: 09 5247670
- Ignore for extract to T&E

 Buttons: Update Dealer, Insert Dealer, Delete Dealer.

Form (b) - Insert Dealer: This form is for inserting a new dealer. It contains the following fields:

- Company Name: [Empty]
- Company Type: LMVD MVD BMVA Private
- Contact Name: [Empty]
- Company Address: [Empty]
- Suburb: [Empty], City: [Empty]
- Region: Auckland/Whangarei
- Postal Address: [Empty]
- Email: [Empty]
- Web Site: [Empty]
- Phone Number: [Empty]
- Phone Number 2: [Empty]
- Phone Number 3: [Empty]
- Fax Number: [Empty]
- Ignore for extract to T&E

 Buttons: Update Dealer, Insert Dealer, Delete Dealer.

Figure 4.6(a) and (b) Example of Task based content display

The two page shots in Figure 4.6 show depending on the task the user wishes to perform. When a user wants to update a dealer information, the page (a) is shown. In the case to insert the information of a “New Dealer” into system, the page (b) will be shown.

The traditional design for the interfaces shown above is to have different versions of files, and just simply copy and paste the part with the same content. This will result the hips of amount of files developed with similar content for similar interfaces.

4.4 Implementation

After design phase, we will start implementing our car site application. We will focus on creating appropriate user interfaces for various devices. The business logic code with data transaction between database and front data presentation will be put in separate tier.

4.4.1 JSP Basics

As described in chapter three chapter three, dynamic HTML and WML document can be generated by CGI scripts, servlets, JavaServer Pages, Active Server Pages, Perl, Tcl, and so forth. We can build dedicated server-side web pages for each different combination of user, user task and device, using Java Server Pages, Active Server Pages, Servlets, PhPs, CGIs, ColdFusion and other technologies and tools [Marsic, 2001a; Han et al 2000; Zarikas et al 2001]. This is currently the “standard” approach.

To implement our car site, we will use JavaServer Page to generate interfaces for each device that might be Web and WAP enabled, and for each type of users and tasks.

JSPs are basically files that combine standard HTML (or XML) and new scripting tags. They provide server-side “scripting” language that contains presentation mark-up, with embedded dynamic processing logic expressed in the Java Programming language. JSPs therefore look somewhat like HTML, but they get translated into Java servlets the first time they are invoked by a client [J2EE Edition].

We have chosen JSPs as our implementation platform for several reasons:

- The main difference between JSP and other technologies to enable rapid development of web-based applications is that JSP is platform-independent.
- JSPs provide a good distributed object-based platform form.
- It can simplify the creation and management of dynamic web pages, by separating content and presentation [Joseph L Weber]. JSPs have feature to separate user interfaces from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

JSP technology needs a web server container. We chose the Tomcat in this sample application as web server.

4.4.2 Database Implementation

There are a wide range of data management strategies for us to choose when implementing object-oriented design. They include relational database (RDBMS), object-oriented database (OODBMS), files (text or binary), XML database. The selection of the data management tools is dependent on the particular application.

Because the intended users of the car site will be internet users and our application is “thin” client, the database needs to be powerful enough to handle high volume data transactions as it might potentially grow to be large in the future as the site becoming more popular. In most cases, the server where the database resides is separate from the web server.

For example, we can choose Interbase as the database server. Interbase has full functionality, similar to MS SQL Server. We can make use of some of the internal data management tools that Interbase provides, such as “triggers”, stored procedures and indices process the data in the data server as fast as possible. This can reduce the data transaction along the network, so as to get better performance.

The database involved in the car site is fairly simple. The following ER diagram shows the part of tables and fields of relational database. The name of the field in tables have special meaning, for example, “slogin” in “dealer” table means the type of this field is “String”, the “sicaryear” in “car” table means the type of this field is “Small Int”.

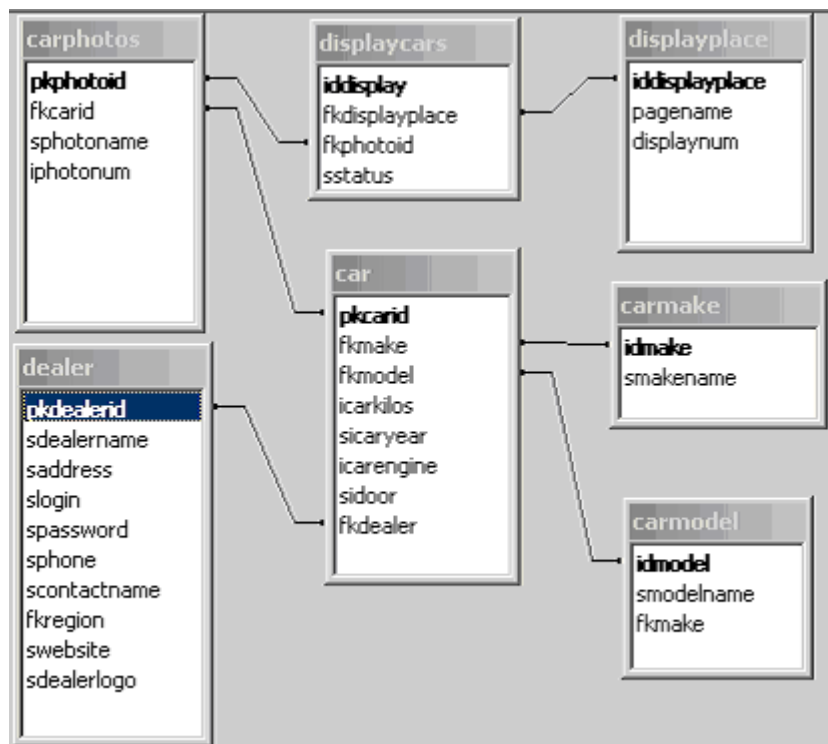


Figure 4.7 Car Site ER diagram

4.4.3 Application Logic Implementation

The backend business logic is provided via Java Beans hosted in the Tomcat server, for access to the database and data transactions. Tomcat is a servlet container with a JSP environment. A servlet container is a runtime shell that manages and invokes servlets on behalf of users [J2EE Edition].

If wml device

If html device

Use

This application consists of a combination of Java Beans that provide the application logic to handle transactions by accessing the database containing the dealer catalog and inventory.

If user1

If user2

If user1

If user2

4.4.3.1 Application Logic Flow

JSP Pages for User1,WML

JSP Pages for User2,WML

JSP Pages for User1,HTML

JSP Pages for User1,HTML

The diagram in Figure 4.8 shows the logic flow regarding how our application starts. When a user uses a device to browse the application, the web server will all require "index.jsp" as first page no matter which device is used. A function bean will check the header information of the device and verify the next page it should be should directed to. It then allows the device client to access a correct version of the JSP document that best fits the user requirement specified in the request header, as several versions exist on the server in our application.

For example, if the device used for browsing this application supports WML, the "WMLHome.jsp" is shown as the first page for the Mobile device. After login, if they are of type "user1", we will show them the JSP pages that are enabled for WML and "user1". The "index.jsp" page does not contain any layout code. It is just used to direct the correct version of code for a particular device. The detailed code example will be shown in the next section.

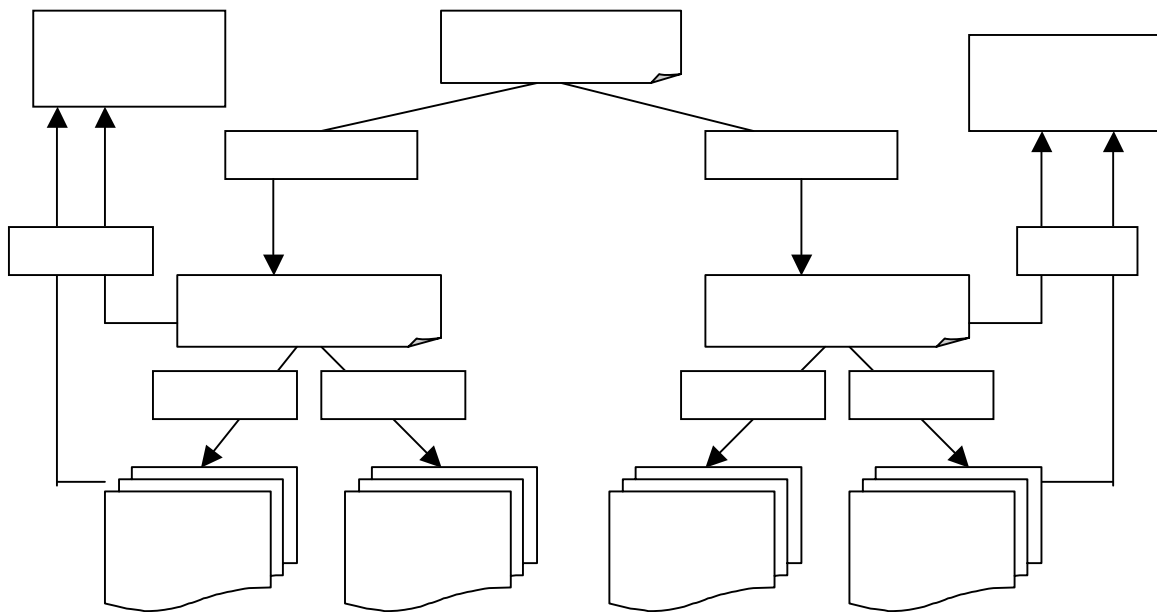


Figure 4.8 Part of Application Logic Flow

All the data is stored in the back-end database. A bean program is written to fetch all the required data from the database and make all the data ready to the front client to fetch. For example, in this application, I have created the “DBConnection” bean as a “data manager” object to handle all the data transaction between the database and front client. And another bean called “Function Bean” will handle all other functions, including the interaction between that client and the system.

When the user browses information from the different device for specific purpose, the data is fetched from database to web server, and properly displayed on the screen by using the appropriate HTML/WML language.

The “FunctionBean” provides a model of the application logic, and is used throughout the process of dealing with device information verification. It is reused in all JSPs to provide user interaction with the business logic. The “DBConnection” Bean encapsulates the necessary database access code.

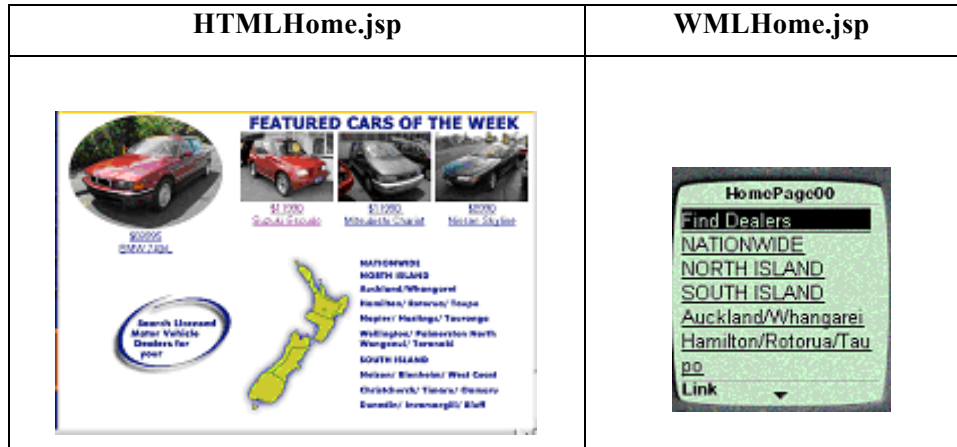


Figure 4. 9 Page dump for homepage for HTML and WML enabled device

On the home page for desktop PC's, people can perform several tasks, including a search for cars through “click a region”, “view featured cars” and they can search a dealer's stocklist by clicking the oval image.

The home page for desktop has a standard screen size and supports rich HTML markup language. We have several areas on the top of a page to show featured cars, but we cannot have the same functions on a mobile device such as mobile phone.

4.4.3.2 Get Device and User Information

- **Obtain Device Value**

The content dynamically generated by the JSP must be in a format supported by the requesting client. For example, for the “Tomcat” server to serve the mobile phone that only supports WML Markup language, the server needs to know what device is browsing the pages, so it can send a WML version rather than HTML.

To identify the client device, the following approach can be used. According to the convention, the value of the User-Agent header field lists the most significant products first. The first product tokens may identify the client category. The categories of a client will either be a product name, like Mozilla and Nokia Development Toolkit, or a product name and its version, such as Mozilla/4.51.

For example, when issuing a request from IE 5.0 and Nokia Toolkit on Windows 2000, the User-Agent field is respectively:

UserAgent: Mozilla/4.0(compatible; MSIE 5.01; Windows NT 5.0)

UserAgent: Nokia-MIT-Browser/3.0

Those tokens can be used as keys to map to the corresponding MIME types. The values of User-Agent header fields can be retrieved by the tag class using the “HttpServletRequest” class and its “getHeader()” method as follows. Based on the type of user agent from the head information, the code will assign the page parameter device value and decide what Markup Language need to be used.

```
HttpServletRequest sr=(HttpServletRequest)pageContext.getRequest();  
String useragent=sr.getHeader("User-Agent");
```

After we know the value user agent, we should be able to know what device is bowering. And the type of Markup language that device support will be known as well.

```
if(device== "html")
    Redirect("/HTMLHome.jsp");
Else
    Redirect("/WMLHome.jsp");
```

- **Obtain User Value**

The user information can be tracked down through “cookies”. After login, if the user can be verified correctly through looking up from back end pre-registered user information from our database, the correct user information can be returned and a session or cookie will be stored in the client.

Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users. We can set cookies using the “setCookie()” function. Cookies are part of the HTTP header, once it’s value has been set, we can get this value from every page of the site by using getCookie() functions.

The cookie setting is just one approach, but there is limitation with the cookie setting approach. For example, Palm VII does not support cookie-based sessions. So that the cookie issue needs to be addressed for each device, as some devices support cookies while others do not.

4.4.4 User Interfaces Implementation

We assume you are familiar with JSP, so that we will not spend more time on how to use JSP.

4.4.4.1 Developing Interfaces for WEB and WAP Enabled Devices

Dynamic WML documents for wireless devices can be easily developed using JSP. Once we know the WML syntax, building WAP applications using JSP can be an easy task.

As we mentioned in chapter three, the basic unit of WML is the card that specifies a single interaction between the user and the user agent. Multiple cards are grouped together in decks. A deck is the topmost element of a WML document. When the user agent receives a deck, it activates only the first card in the deck.[Queay H.Mahmoud]. Therefore, a deck has to be a valid XML document, which implies that a WML documents should start with the standard XML header and the reference to the WML DTD.

Java Server Pages let us embed Java statements within HTML documents, and also WML documents, and create dynamic WML documents. Most of the preceding code uses pure WML tags, with the exception of the line:

```
Response.setContentType("text/vnd.wap.wml");
```

This line ensures that the correct MIME type is set for the WML document, and makes sure that the WML enabled browser is able to parse the content.

Since we are using JSP technology, we are primarily concerned with JSP here. The sample application for WML enabled devices will be a combination of pages with WML decks and dynamic JSP pages that will enable all functions on those devices to work the same as on a desktop.

Let's start by creating the WML deck with only one card, which will serve as the main "index" page. If someone browses the web site from a mobile phone, the "index.jsp" page is first invoked, and after detecting the type of device, the following page can be called. "wmllogin.jsp" is displayed, which would display the following screen.

The code shown in Figure 4.10 has ignored the embedded function code such as login id and password verification and only shows the presentation part of the code. Window (a) and (b) are two page shots for the mobile device. After entering the correct the login id and password, user then presses the relevant key on the device. The window (b) will then show. The “Submit” button is functionally identical to the “Login” button shown on the window (b).

The login information will be passed to a bean that will execute the JDBC connection to the database Server for verifying whether the login id and password are correct.


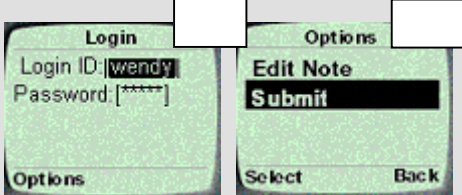
HTMLlogin.jsp	WMLlogin.jsp
<pre><html> <head></head> <body> <form method="post" action="" > Enter Your Login ID and Password Login ID: <input name="userid" type="text" value="">
Password: <input name="password" type="password" value="" > </Form> </body> </html></pre>	<pre><% response.setContentType("text/vnd.wap.wml"); %> <?xml version="1.0"?> <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml"> <card id="card1" title="Log" newcontext="true"> <p> Login ID: <input name="userid" type="text" value=""/>
Password: <input name="password" type="password" value=""/> </card> </wml></pre>
	

Figure 4.10 Code Example and page dumps for login page

4.4.4.2 Examples of User Interfaces

In this section, we will present the user interfaces that we have developed for different devices. We assume that users will use one of three type of devices. For each device, we need to write three customized interfaces to suit onto the screen of these devices.

Example One – “Search Car” Interface

The following interface provides a form that allows the user to specify the search criteria. For example, a user might want to search all cars with “Toyota” make and “Camry” model. The interfaces capture the information that user has specified, pass to a “FunctionBean” for processing, and the function in “DBConnection” bean will connect to backend database and retrieve the car information, then send it to the browser for display.

When designing the search criteria input for the mobile device, we decided to “chop off” one or more criteria to provide better layout. And we did not display the information that the user has selected previously, for example, “NATIONWIDE Search”. The information that the user has selected from a previous page is shown on the desktop browser, but not on other devices.

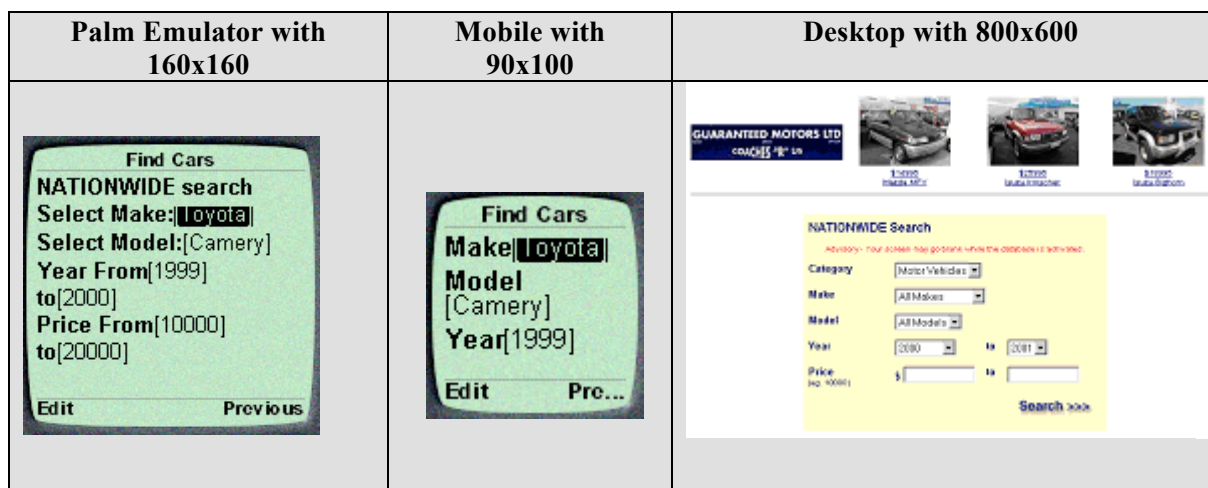


Figure 4.11 Search Car Interface on Various devices

Example Two – “Search Result” Interfaces

Now let us look at the search result interfaces shown in Figure 4.12. After a user specifies certain criteria by selecting or entering values into above interfaces through different devices, the interfaces in Figure 4.12 display the result list of cars. Each window only can display certain number of cars with respect of window size, so the user then needs to use the “next” button to get the next list of results.

We have only shown the interface on three devices with known window size. If there is a need to display our interface on a window 400x300, we need to write a new version of interface and determine how many cars can fit onto that interface.




Palm Emulator with 160x160 can display three cars per screen	Mobile with 90x100 can display one car per screen	Desktop with 800x600 can display 15 cars per screen																																																																																
		 <table border="1"> <thead> <tr> <th>Status</th> <th>Year</th> <th>Vehicle</th> <th>Kms</th> <th>Price</th> </tr> </thead> <tbody> <tr><td></td><td>1999</td><td>Audi A8 Quattro</td><td>90000</td><td>\$45999</td></tr> <tr><td></td><td>1999</td><td>BMW 320i</td><td>9000</td><td>\$33995</td></tr> <tr><td></td><td>1999</td><td>BMW 320i</td><td>12200</td><td>\$54590</td></tr> <tr><td></td><td>1999</td><td>BMW 320i</td><td>8800</td><td>\$51590</td></tr> <tr><td>Arriving</td><td>1999</td><td>BMW 320i</td><td>52000</td><td>\$23999</td></tr> <tr><td></td><td>2000</td><td>BMW 320i ASE</td><td>2716</td><td>\$90000</td></tr> <tr><td></td><td>2000</td><td>BMW 320i ASE</td><td>5000</td><td>\$90000</td></tr> <tr><td></td><td>2000</td><td>BMW 320i Coup</td><td>6500</td><td>\$91590</td></tr> <tr><td></td><td>1999</td><td>BMW 320i SE</td><td>41070</td><td>\$69990</td></tr> <tr><td></td><td>2000</td><td>BMW 320i Touring</td><td>9000</td><td>\$79990</td></tr> <tr><td></td><td>2000</td><td>Chrysler Neon</td><td>15001</td><td>\$21995</td></tr> <tr><td></td><td>1999</td><td>Chrysler Valiant</td><td>34000</td><td>\$31995</td></tr> <tr><td></td><td>2000</td><td>Chrysler Voyager SE</td><td>500</td><td>\$45995</td></tr> <tr><td></td><td>1999</td><td>Citroen Xsara SX</td><td>45095</td><td>\$25995</td></tr> <tr><td>NTW</td><td>2001</td><td>Daewoo Lanos SE</td><td></td><td>\$19495</td></tr> </tbody> </table>	Status	Year	Vehicle	Kms	Price		1999	Audi A8 Quattro	90000	\$45999		1999	BMW 320i	9000	\$33995		1999	BMW 320i	12200	\$54590		1999	BMW 320i	8800	\$51590	Arriving	1999	BMW 320i	52000	\$23999		2000	BMW 320i ASE	2716	\$90000		2000	BMW 320i ASE	5000	\$90000		2000	BMW 320i Coup	6500	\$91590		1999	BMW 320i SE	41070	\$69990		2000	BMW 320i Touring	9000	\$79990		2000	Chrysler Neon	15001	\$21995		1999	Chrysler Valiant	34000	\$31995		2000	Chrysler Voyager SE	500	\$45995		1999	Citroen Xsara SX	45095	\$25995	NTW	2001	Daewoo Lanos SE		\$19495
Status	Year	Vehicle	Kms	Price																																																																														
	1999	Audi A8 Quattro	90000	\$45999																																																																														
	1999	BMW 320i	9000	\$33995																																																																														
	1999	BMW 320i	12200	\$54590																																																																														
	1999	BMW 320i	8800	\$51590																																																																														
Arriving	1999	BMW 320i	52000	\$23999																																																																														
	2000	BMW 320i ASE	2716	\$90000																																																																														
	2000	BMW 320i ASE	5000	\$90000																																																																														
	2000	BMW 320i Coup	6500	\$91590																																																																														
	1999	BMW 320i SE	41070	\$69990																																																																														
	2000	BMW 320i Touring	9000	\$79990																																																																														
	2000	Chrysler Neon	15001	\$21995																																																																														
	1999	Chrysler Valiant	34000	\$31995																																																																														
	2000	Chrysler Voyager SE	500	\$45995																																																																														
	1999	Citroen Xsara SX	45095	\$25995																																																																														
NTW	2001	Daewoo Lanos SE		\$19495																																																																														

Figure 4.12 Display Search Result Interface on Various Devices

Example Three – “Search Dealer” Interfaces

Figure 4.13 provides three interfaces allowing user to select a particular dealer. In the desktop interface, we can see some of the graphic designs, such as a “gif” formatted image on the top of the window with the word “Dealers”, and a “search” button for users to press to proceed with a search. We would not have the similar elements on the mobile device.




Palm Emulator with 160x160	Mobile with 90x100	Desktop with 800x600
		

Figure 4.13 Search Dealer Interface on various devices

Example Four – “Dealer List” Interface

The dealer list interfaces display a list of dealers based on the region and key word the user has selected from the interfaces shown in Figure 4.13.

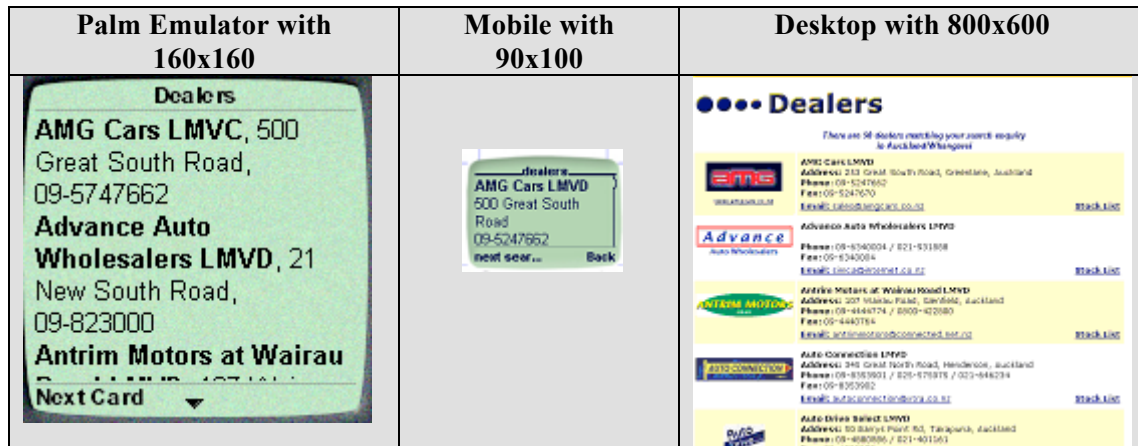


Figure 4.14 Dealer List Interfaces on various devices

From the above examples, we can see the different interfaces need to be designed based on the screen size limitation of various devices.

- Mobile devices can't display the graphic dealer logo as seen on the desktop web browser.
- Mobile devices can't display information for too many dealers on each window, not just because the user has to view the information by using the key to scroll the window, but also due to their memory limitation.
- On a desktop browser, we can see that a website address is displayed for each dealer. Mobile devices have no problem displaying them, but can't have links to related dealer web site. Because those web site servers may not have “WML” version to support mobile device.

4.4.4.3 User and Task-based Content Display

In this section, we will explore the user and task content display using the traditional approach. Traditionally, a number of interfaces will need to be developed for each combination of user and task. Most parts of similar interfaces will need to be copied and pasted in each file, then modified as needed. We will end up with a lot of files with similar content.

Apart from the approach we mentioned above, we can improve a little by adding conditional constructs to the screens for user and to some degree user task adaptations, somewhat reducing the total number of screens to build.

For user-based content display, we need to identify the user. Based on identification of the user, the page will determine what content needs to be shown and what to hide.

A most common approach adopted by developers is to get the value of the user using the approach explained in section 4.3.3.2 for a particular page, then use several “if/else” statements to perform differently based on the value, in other words, to generate different layouts.

Code example

```
if(user== "dealer"){  
    showDealerContent();  
}else{  
    showUserContent();  
}
```

The content shown or hidden from displaying can also be determined by a particular task that the user performs. For example, the screen (1) in Figure 4.15 shows after a sales representative logs from login page successfully. This page displays a list of dealers with several link entries on it for the user to perform tasks.

After the user clicks the links that indicates the id of a dealer on the left side of screen (1), the screen (2) shows, and the screen (3) will show if the user clicks the “Register new dealer” button.

Screen (2) and (3) have similar interfaces. Screen (2) allows the user to update the information about a dealer. It displays the current information we stored in database about that dealer.

Task="update"

Task="insert"

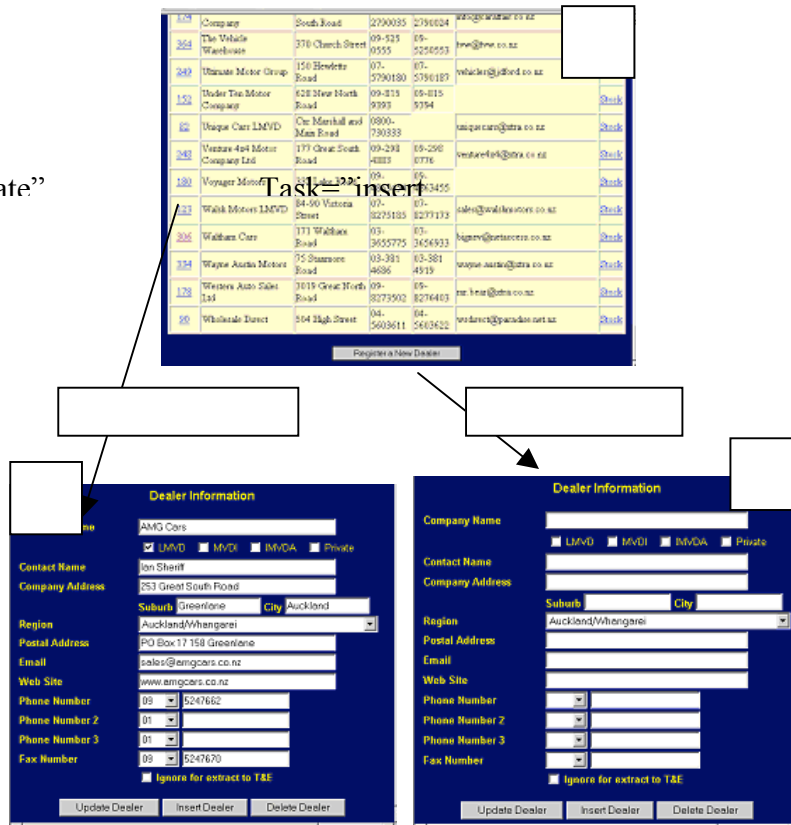


Figure 4.15 Example of task based content display

In this case, the page discriminates by verifying different parameters passed to this page. For example, the value of parameter “task” passed to screen (2) and (3) in Figure 4.15 from different source is different. Passing different parameters to the same page, will lead to different layouts allowing different tasks to be shown or performed by users.

```

<?String task=request.getParameter("task")?>

<% if(task== "update"){ %>

    <!--Content -- >

<}%>
<% else if(task== "insert"){ %>

    <!--Content -- >

<% } %>

```

Figure 4.16 Code Example

The code example in Figure 4.16 shows the general approach that people normally use to solve user or task based content process. In the real world, things are not just as the above

example. We could have a much more complicated situation which would end up with a lot of “if/else” statements embedded in the middle of a page. If the static code between if and else is very large, this will make the page very hard to maintain.

There is another situation when we might display the similar content in a page. If we show a list of cars, normally we would use the following loop statement.

```
<%for (i=0;i<num;i++) {%>
    ...html content
<%}%>
```

Figure 4.17 Code Example

The embedded code will become more complicated if we mix the above two cases. This kind of code is not easily moved to the logic tier, and normally stays in the page.

However, for even small numbers of different users and user tasks the number of adaptations required for this approach makes screen implementation logic very complex and hard to maintain [van der Donckt, 2001; Grundy and Hosking 2001]. Each different device that may use the same screen still needs a dedicated server-side implementation [Fox et al, 1998; Marsic, 2001a] due to different mark-up, size, colour etc characteristics.

So there is a need to find an approach to solve above problems and make the code clearer and more maintainable. We will discuss this further in the next chapter.

4.5 Running and Deploying the Application

To assist in developing WAP applications, I have used “Nokia Mobile Internet Toolkit” as development tools to do testing. The major advantage with this tool is that it has a “device setting” function to allow the various screen sizes to be set. Nokia Mobile Internet Toolkit is available from www.forum.nokia.com. This is free of charge, but requires registration.

To install a Web Server such as “Tomcat”, the steps are

- Download the file from <http://jakarta.apache.org/downloads/binindex.html>

- Unzip the file into a directory and create new subdirectory named “tomcat”
- Put all JSPs into the jsp directory and other Java Classes into classes directory.

4.6 Summary

In this chapter, I have provided a demonstration of how to develop Web and WAP applications in Java JSP using a traditional approach by modelling an online car sale web site. The uncommon part of this application is that the client types are not just desktop computers, they include other devices like mobile phone and PDA.

To resolve the requirements of these different devices, we have specified a standard approach to solve the problem. We have also noticed some other problems, such as the need to develop a large number of single interfaces for each device and user or task.

In the next chapter, we will discuss in more detail the main problems the standard approach presents and provide the motivation behind the need to develop improved approaches to user interface adaptation which arose from experience developing the car site using traditional approaches.

Chapter 5 - Adaptable User Interface Technology

5.1 Introduction

In this chapter, we present

- The motivation for exploring adaptable user interface approaches.
- The objectives that we wish to achieve
- The approaches to developing adaptable interfaces for various human devices.
- Some sample applications to access real-world data.

We will introduce the Adaptable User Interface Technology (AUIT), a new approach to building adaptable, thin-client user interface solutions for web-based information systems. We present the key design features of AUIT in this chapter, a prototype and implementation in the following chapters.

5.2 Motivation for Developing an Adaptable Approach

In chapter four, we reviewed how to develop WEB/WAP-based application for various devices using a traditional approach. The main advantage with the traditional approach is that it is very flexible. The developer can write specific programs or pages for particular devices. This methodology is good for writing specific applications based on the particular features of a device.

Let us discuss the main disadvantages of the traditional approach to understand the need for an adaptable approach to developing user interfaces in more detail.

5.2.1 Need to Develop Large Number of Interfaces

From the experience of building user interfaces for our car site using traditional approach, we can see that the number of interfaces required to satisfy our multi-platform & user-task requirements is very large. This presents major problems both in development time and

ongoing maintenance. - for M different information system screens and N different user, user task and device combinations, we have to build and then maintain M*N screens [van der Donckt et al, 2001].

5.2.2 Hard coded for Each JSP

Developers will need to spend more development time. They have more chance to write a lot of mindless coding that is more suitable for a machine than a human being. In such situations, developers might just need to simply cut and paste of the existing code for other devices and modify small part of code, and reuse them.

5.2.3 Fast Growing Devices

As new devices and display technologies emerge, developers will need to add new code for each new device. This will increase a workload for developers.

From the analysis of the traditional approach, we can see that although traditional approach could be the easier option in some cases, such as some small Web sites, it will require additional code for a page (in our case, JSP or servlet) every time a new device has to be supported. This is because wireless devices vary widely in a lot of ways.

5.2.4 Bad Code Reusable Ability

The back end access calls are repeated in each JSP. The code for display the user interface will be duplicated with each device. It makes a web site difficult to read and maintain. The code is also less maintainable.

A lot of embedded code is needed for user and task based user interface. For example, the following JSP code will handle the content generation in terms of the different users in traditional approach.

```
<%if(user=="manager") {%>
  <!-- Content for manager -- >
<%}else if(user=="staff"){%>
  <!-- Content for staff -- >
<%}%>
```

However, a mass of conditionals can make a JSP verbose and hard to maintain. Even the simple “if/else” example above already requires a few seconds to comprehend. If lengthy blocks of HTML code break up the conditional statements, or if there was a longer chain of “if-else” statements, it could quickly become unreadable.

Therefore we can see that to create user and task based content, the code often consists of a mish-mash of program code in scriptlets and HTML/WML markup.

5.2.5 Lack of Consistency

Because the interfaces developed for each device are completely different, this might lead to the interface among devices being inconsistent.

5.3 System Requirements and Specification

Using an adaptable approach offers significant development time and workload reductions because the developer needs only to create a single system which will suit any device.

With these goals in mind, we need to determine the requirements of our system. These requirements depend mainly on the web developer’s perspective and demand. Using this approach, we will build a package that can be adopted by developers as tools, or as a basic concept of approach to provide an alternative to the traditional approach which resolves it’s inherent problems.

5.3.1 Functional Requirements

In this section, we will describe the functional requirements that the system will be providing for developers. The following specifies the detailed functional requirements that the system should meet.

5.3.1.1 Write Once, Run Everywhere

The system should allow developers to create applications that have a “write-once, display on any device” capability, simplifying the process of differences between devices, such as Markup Language and different screen sizes.

- **Dynamic Markup Language Generation**

This is one of the main requirements. Because different devices may support different Markup languages, the system should provide ability to generate markup language dynamically to serve any predicted device that a user may used to browse the application.

Adaptation is performed when the client device requests service-and the adaptation system will detect the device, decide the markup languages and present the layout.

- **Adaptation to various screen sizes**

There are many human-computer devices existing in today’s market and more will come to market soon. They all have different screen sizes to display the information. In a non-adaptable approach, to make a web application support wireless devices, we need to rewrite another set of code to make to content fit well on the screen. Our Adaptable approach will do this job on the run time automatically, generating appropriate content for each specific device.

5.3.1.2 Adaptation to User and User preferences

Some user interfaces and/or elements suit some users but not others, based on the particular user’s role or subtask being performed. Most of the adaptable systems set the user and user preference adaptation ability as their goal. Individual users' needs and preferences may change as they use a software system. [Kules 00] The adaptable system monitors the user's activity pattern and automatically adjusts the interface or content provided by the system to accommodate user differences as well as changes in preferences.

Users may specify preferences about which elements or alternative interfaces they want to use, default values and constants. . For example, in this particular adaptable system, the user can prefer to use a special window size to view the interface instead of using the

system default window size. For instance, they might like to use window size with 600x200 pixels to view the content using normal desktop browsers like IE or Netscape.

User and user preference based adaptable techniques provide a way to optimize a user interface for individual users.

5.3.1.3 Adaptation to Tasks that Users Perform

This is also common target of adaptable user interfaces. The user can easily control the layout of the task performed. This is similar with the user-based requirement.

Within the traditional approach outlined in chapter 4, we used simple “if/else” statements to control the content that needs to be displayed. With the adaptable approach, we can eliminate that messy embedded code in the page and achieve the task based user interface adaptation more easily.

5.3.1.4 Adaptation to Client Devices

With a potentially unlimited amount of RAM and hard disk space available on today’s desktop machines, developers are less constrained in the software they design. Of course, they don’t have that luxury with mobile applications. Developers have to remember that mobile phones have small screens with low resolution, limited memory and processing power and less bandwidth than desktop systems. Some content is not suitable display on the some wireless devices; for example, large image files.

The adaptable interface should control what elements are displayed on a given device based on window size or memory availability. These parameters are set in the first instance by the developer.

5.3.1.5 Adaptation on Existing Client-Server Architecture

Web developers are familiar with the existing Client-Server Architecture during development of web application. Our system should make use as much as of the existing architectures that developers are familiar with.

The developer can make use of their original web development knowledge (such as architecture) to develop the new web site without having to learn any new knowledge.

5.3.1.6 Configuration Capability

The system should provide a set of interfaces that can be used to catch information about that device, such as user-agent information about the device.

- Each device has its own specific characteristics, such as width and height of screen.
- The size of elements displayed on a specific device, compared with the default elements size, such as the catch radio value of default text size on a PDA over the default text size on PC browser.

If a web site has more than one type of user, and requires different layouts for different users, the related user information also needs to be captured through the configuration interface.

Therefore, configuration capability of a system is important. This provides a flexible way for developers to cater for many devices and users.

5.3.2 Non-functional Requirements

In this section, we will explore non-functional requirements that our system should provide.

5.3.2.1 Easy to Use

The system that we are going to develop should provide “easy to use” features for developers. This requirement comprises of two parts based on two groups of end users who might use our adaptable approach, developer and application end-users.

From a developer perspective, their role is to develop the application for the end user, using our adaptable approach. It should be easy to use by developers. When they use our

package for implementing our adaptable approach to develop a web site, they should not spend too much time learning the system.

From end user point of view, their concern will be whether the application is easy to use, no matter what approach or technology was used to develop this application. The interface and the functionalities of application developed using adaptable approach should make no difference from their perspective.

5.3.2.2 Easy Read and Maintain

Normal standard technologies, like Javaserer Pages, ASP, PHP, etc, have an ability to separate code from content. All embedded code can be split from the page and placed in a class module. This allows graphic designers to make the site look nice, while the programmers can take care of the coding.

Some of the adaptable approaches can provide good results superficially, but with very bad, illegible and hard to maintain code. To avoid this, our page should look like an HTML page as much as possible with minimal dynamic content, rather than a program with embedded markup or even a balance of scriptlets and markup.

Graphic designers normally only have knowledge of HTML so if our page looks like HTML, they will be able to contribute their skill to maintain and enhance presentation on a site. The user interface on today's web sites can be very complex. Unless the page generation approach in use is understandable by page graphic designers, it will be impossible for them to take part in graphic design on a web site.

5.3.2.3 Performance

When people use a web-based application, the main expectation is a low response time required to display each interface. Good performance is a very important aspect of the overall web application.

The application should provide satisfactory performance when developers develop a web application using our adaptable approach.

5.3.2.4 Programming Productivity

The main advantage of using an adaptable approach is increasing programming productivity. This is one motivation for us to use an adaptable approach to develop a web based application for various human devices. The ability to develop and then deploy applications as effectively and as quickly as possible is important.

5.3.2.5 Extensibility

Because the holistic environment is not static there is a requirement for most projects to have some degree of extensibility designed in. The ability to have a project or a system extensible is key to the success of developing for today's system.

Our system should be able to be extended by other developers. It needs to be easy for them to use our idea by extending our system.

5.4 System Analysis

Achieving these requirements can be complicated, because of the variety of technologies and standards that have been developed over the years, requiring highly developed skill sets. The acquiring of which and “keeping up” of those skill sets is a problem in itself. Moreover, the rapid pace of change in ‘standards’ themselves poses significant challenges to ensuring efficient meshing of technologies. [J2EE Edition]

With the above-specified system requirements, we need to do an analysis for the system first. Because the requirements might be far more than we could do in reality, we will start by making some reasonable assumptions, and then develop system within this simplified world. Then we can then progressively add to our basic system to meet more complicated requirement.

5.4.1 Adaptation Ability to Various Markup Languages

The following table lists some of Markup languages that diverse devices can support.

Client	Markup Language
PC-based browser	HTML, DHTML, XHTML
PDA	WML, XHTML
Mobile phone	WML, XHTML
Landline phone	VoiceXML
Server application-specific	XML languages

Figure 5.1 The Markup Language and Devices

For our system, we will assume that various devices could use two main languages, HTML and WML. Even though devices in today's market could support a wider range of languages and standards, we can always allow our system to adopt more markup languages later. Once it can support two markup languages, it should be no problem to add up more. We can even develop some tools for developers to add any syntax for new markup languages to make the system support them.

5.4.2 Screen Size and Page Content

Most of the time, existing user interfaces are developed by UI designers with one context of screen size in their mind. For example, most of web developers design pages for a web site with the constraint of screen display size 800x600.

With our approach, we propose to have functions to identify the users and devices that are pre-stored within our database using a configuration engine. For example, if a user wants to use IE to browse the content of a web site using a window size of 400x200 pixels as his/her preferred screen size, the content will be displayed on the window with that window size constraint.

Therefore, we will assume that the designer has designed the interface based on the content of a page and not the constraint of the window size. In other words, they do not design their interface based on a particular screen size. For example, a web page can display unlimited content on each page and users can view the content normally by scrolling the screen right and down. To make users with restricted screen size feel more comfortable, we

can create an internal algorithm which separates the content into several windows without interrupting the flow or overall display of the content. We will explain the detailed algorithm and the result in a later chapter.

5.4.3 Cookie Problems

Cookies are useful for maintaining state and keeping track of users' sessions. Although cookies are part of the WAP specification, they are not yet implemented by all WAP browsers. The Nokia 7110 does not yet support cookies. Phone.com's "UPSimulator" however, does support them.

To allow our system to work on any device and retain the capability to vary the content for different users, we will assume that all the devices that will use our system support cookies.

With these assumptions in mind, I decided to focus on activities that required the minimum amount of input, complexity and logic flow. I also made the decision to first develop a simple system with only partial functionality requirements fulfilled; then expand it with more functions.

5.5 Design and Algorithm

From our practical experience of developing a car site using the traditional approach, as specified in chapter four, the above analysis of the adaptable system and with the system requirements and objectives in mind, we now move into the design phase. This is one of the most challenging and creative parts of system development.

We will develop a new approach to building adaptive, multi-device, thin-client user interfaces for web-based information systems to meet our requirements. We intend to design and create a set of elements to make up adaptable user interfaces. These elements are tags that are specified using a device-independent mark-up language describing screen elements and layout along with any required dynamic content. Screen element descriptions

PDA

Mobile

Desktop

may include annotations indicating which user(s) and user task(s) the elements are relevant to.

AUIT Pages
Business Logic

Database

We call this Adaptive User Interface Technology (AUIT). The interfaces developed using AUIT elements are called AUIT pages.

5.5.1 System Architecture Design

Software architecture is used to determine the overall design of the system. In designing the architecture, we need to consider the following issues.

- Which programs we are going to use.
- The environments and machines these programs will run on.
- How these machines are going to be networked.

In order to develop a new approach to building adaptive, multi-device user interfaces for web-based information systems, we need to look at the kinds of architecture that can be applied. We have described a general 3-tier architecture used to develop the thin-client information system in chapter four.

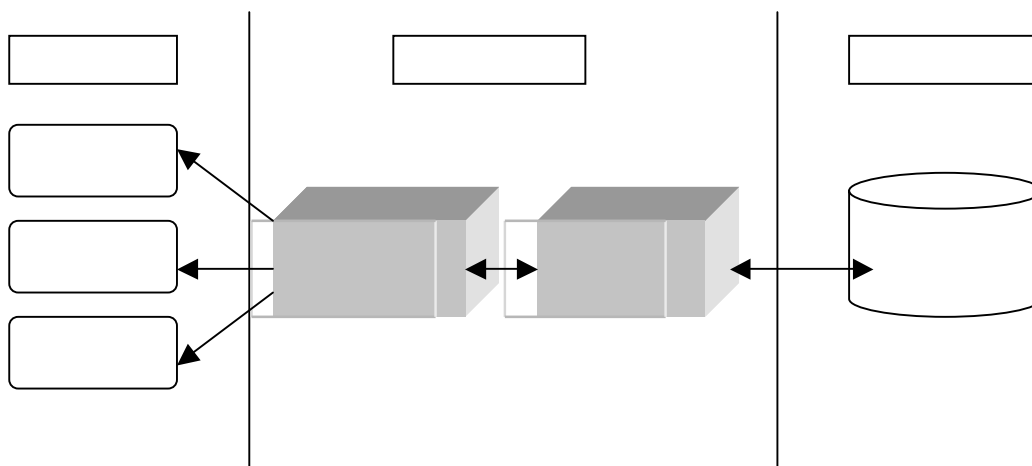


Figure 5.2 General Architecture Using AUIT System

Our AUIT system adopts the above four-tier software architecture illustrated in Figure 5.2. This is based on the architecture used for the traditional approach with some modification. Clients can be desktop and laptop PCs running a standard HTML based web-browser;

mobiles or PDAs using WML-based browser, or mobile devices like pagers and WAP phones, providing very small screen WML-based displays.

AUIT pages sit between “client” and “business logic”. The web layer retrieves the data and formats it for display. This separation of business logic from the AUIT presentation pages adds flexibility to the design of the application. Multiple-user interfaces can be built and deployed without ever changing the business logic provided the business logic presents a clearly defined interface to the presentation layer.

Compare this architecture with the one we have specified in the traditional approach. Excepting the new tier (AUIT pages), the other tiers should perform the same functionalities as we described in the traditional approach. This architecture has the advantage of only needing to use the new approach to develop the interface. We can make use of any previous knowledge with developing dynamic code for handling business logic (such as data transaction between interface and the database) without needing to reengineer the business logic.

5.5.2 Designing AUIT Elements

With the architecture designed, we need to consider how we will create the AUIT pages. Our adaptable application should be made up of AUIT pages to serve the client device. An AUIT page should consist of AUIT elements. Therefore we need to extract the common elements from the two different languages and create a list of high-level elements to achieve our purposes. These elements are not HTML tags, nor WML tags, but a high level language that used to write our AUIT page.

5.5.2.1 Document Structure Elements

From Figure 5.3, we can see that each document, such as WML or HTML used to render use interface for thin-client applications have a basic document structure. Thus we need to have one or more elements to specify the basic structure of our AUIT page.

A valid WML deck is also a valid XML document, the same as HTML. Its basic structure is shown on the left of the following table. An HTML document is composed of three parts and a WML document is composed of two.

WML document	HTML document
<pre><?xml version="1.0"?> <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml"> <wml> <card id="First_Card" title="First Card"> <p> ...all other elements... </p> </card> </wml></pre>	<pre><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd"> <html> <head> ...head information </head> <body> ...all other elements... </body> </html></pre>

Figure 5.3 Document Structures of WML and HTML

We will use an element called “Template” to specify the basic structure of our AUIT page. The document structure element should provide the main structure required by both an HTML and a WML document. An AUIT page should start with <template> elements, end with </template> elements and has a set of other type of elements in between.

5.5.2.2 Generic UI Elements

Each interface is made up of a set of UI elements. Now we need to design a set of UI elements that can make up the user interface specified on an AUIT page. First of all, we should create a list of UI elements that can be used to generate proper markup language for a given device to render the content on the page. From the standard DTD of HTML and WML, we can extract some common elements.

In the following diagram, we have designed a set of UI elements such as user input elements to provide a mechanism for capturing user input, regardless of the markup language that the device supports. For example, a “Label” element can display text and a “Layout” element formats the text.

We can see that the elements and their attributes have similar names as used in HTML. We have followed the basic methodology established for HTML for the purpose of easy adoption by developers.

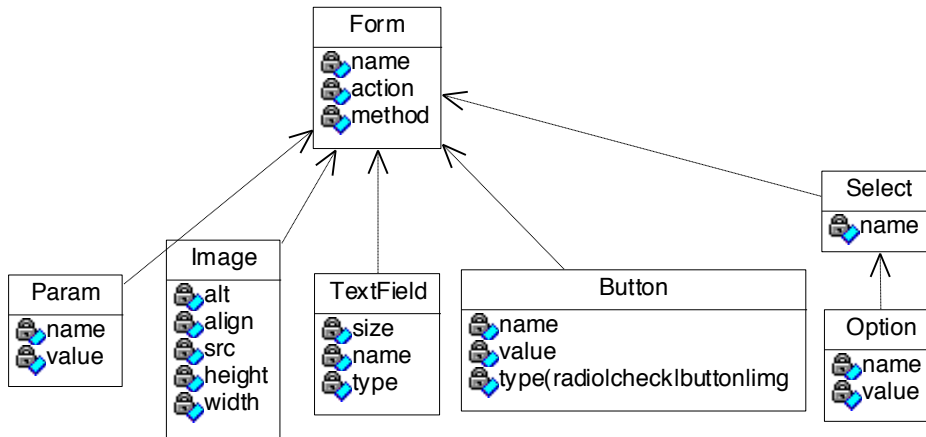


Figure 5.4 Form and user input elements



Figure 5.5 Text and text format elements

Figure 5.6 Navigation element

The following provides detailed explanations.

- Generic form elements:

The form elements in HTML and WML are very different, as is the way they pass the parameters described in chapter two. We have designed a form element that generates different formats for different devices.

Some AUIT form elements such as the “textfield”, “button”, “checkbox” and so on, are also designed for generating relevant UI elements for both markup languages.

- Generic label and layout element

These elements are used for describing the layout for the text display. The HTML provides much more elements and attributes that may be used for visual formatting than WML, such as <H1>, , <DIV>, stylesheet, etc; which WML does not provide. We use a <Layout> element to combine the common parts of HTML and WML to format the text displayed on a device.

- Generic navigation elements:

The navigation through a web site is achieved by a link that is a connection from one Web resource to another. Both WML and HTML provide navigation tags, but they have some differences. The generic elements integrated these differences; provide a “link” element to achieve this purpose.

5.5.2.3 UI Control Elements

We assume that on each device, the designer will want to accomplish the same amount of tasks (or subsets of the tasks) for each page. For example, a page has ten UI elements on it and the amount of tasks that user can perform is three. So any device that used to browse this page should have as a maximum, those ten UI elements and those three tasks.

We have used three elements, user, task and device, to achieve different type of UI control. The basic idea of how these elements work is that they will hide part of an AUIT page layout based on the type of user, the type of device. etc.

The UI control elements shown in Figure 5.7 are responsible for control of the UI elements. They decide if the UI elements will be displayed (such as “Task”, “Device” and “User”) and how they will be displayed (such as “Iterator” and “Table”). The following diagram shows the elements and their attributes which are used for different occasions.

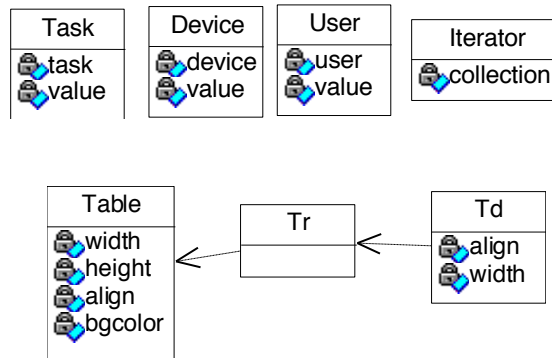


Figure 5.7 UI control elements

From Figure 5.7, we can see that Task, Device and User elements have a similar structure and application, but control the layout by showing or hiding through checking different conditions. For example, Task elements controlling content between a start tag and end tag will be shown if a user performs a certain task. Otherwise, it will not be shown.

There are also some other elements provided within this project, like an “Iterator”, which is used to loop the same content.

5.5.3 Adaptation to Various Screen Sizes

So far, we have designed a set of basic elements for our AUIT page to create an adaptable user interface. We have achieved a number of tasks specified in our requirement specification. Now we need to design an approach to have our AUIT page adapt to any size of screen.

5.5.3.1 Approaches

In typical web based application, the interface designer normally designs an interface based on the window size of a desktop PC. For example, a web page should be produced for an 800x600 display. Our approach however, will allow the AUIT page to display appropriate content on any screen, irrespective of size.

Based on our specified requirements, we will need to intelligently transform a given user interface from one context to another one; thus, if needed, multiple user interfaces can be generated simultaneously. Our approach to achieve this is to have an internal

transformation that will perform “splitting” of screens into multiple screens when all items in a form cannot be sensibly displayed in one go on the display device.

For example, when text will over-fill the display device screen, text to the right and bottom over-filling the screen is moved to separate screens, linked by hypertext links. This means the user does not need to scroll either horizontally or vertically to reach over-flowing information, producing a more easy to use interface across all devices.

Mobile devices that have small window size can move the overflow content to the next card. The user can choose to view the content on the other card by clicking on the relevant link.

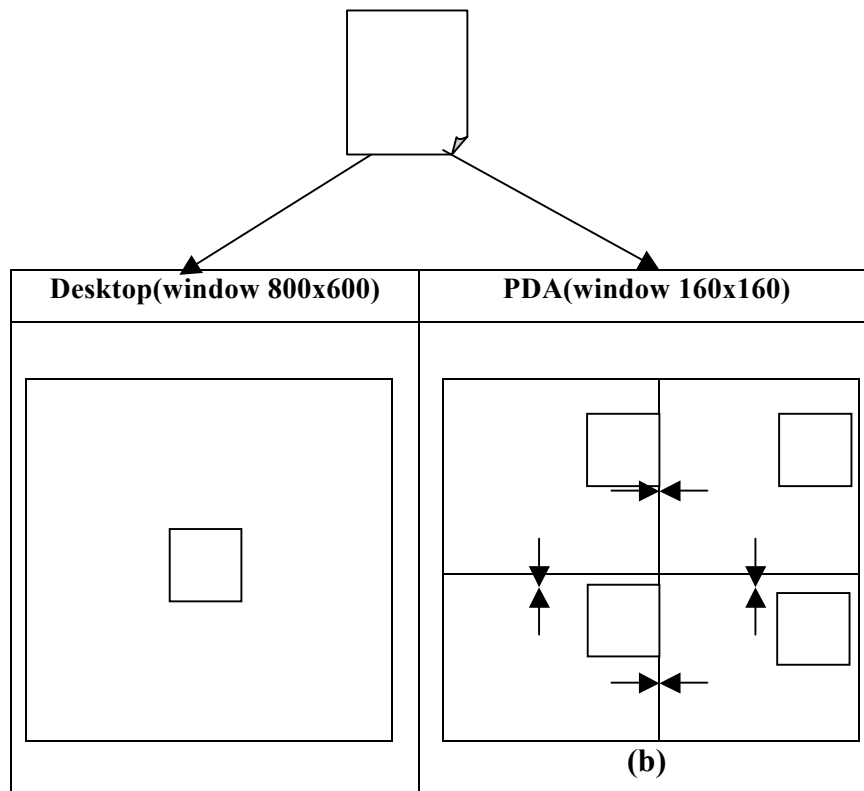


Figure 5.8 Example about screen splitting

For example, when a AUIT page shown in Figure 5.8 is running at server, it will generate the page (a) if a user uses a desktop browser with window size 800x600 pixels to view. It will generate the screen (1) if the user is on a PDA with a window size 160x160 pixels and the user can click the right arrow link on screen (1) to view content on screen (2).

Therefore the entire content can be viewed by using the navigation arrow link. For a PDA device, after an AUIT tag is processed, it generates output with four parts, but only displays the first part, caching the rest in a buffer instead of outputting them all at once.

We designed a built-in algorithm to generate navigation links automatically if there is need to separate the content of a page into several pages. These links are not specified by the users, but exist in the generated HTML or WML code to direct the web user to view the entire content of a page. The direction of the link is also generated automatically.

Another example of this process is outlined in Figure 5.9. Here a PDA requests a screen that is too big for it to display, so the AUIT tag output is grouped into multiple screens. The PDA gets the first screen to display and the user can click on the Right and Down links to get the other data. Some fields can be repeated in following screens (e.g. the first column of the table in this example) if the user needs to see them on each screen.

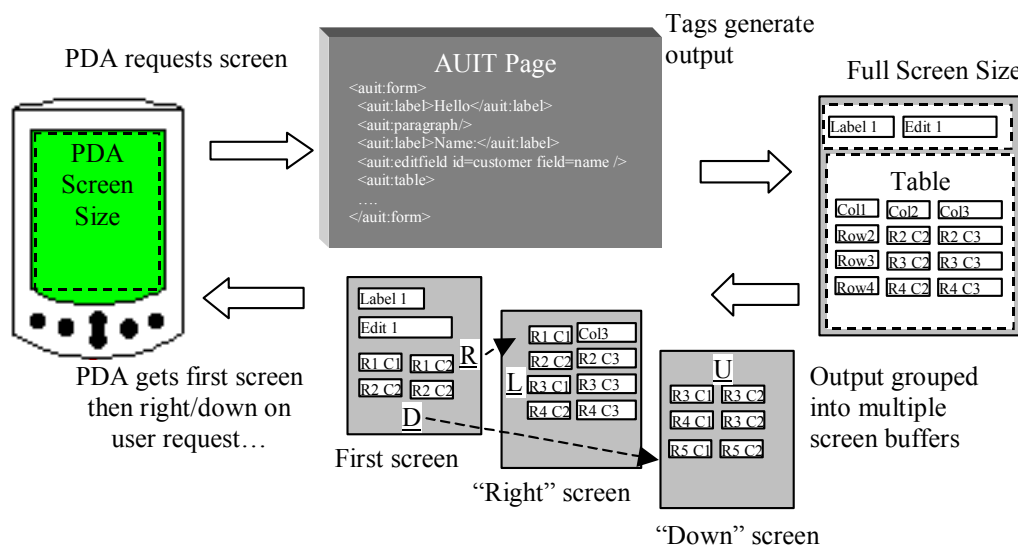


Figure 5.9 Example of screen splitting

5.5.3.2 Design Elements to Perform Screen “Splitting”

So far we have defined some AUIT elements that can be adapted to generate Markup language for various devices. No matter what programming language is supported to render a user interface, we can see that each interface is actually composed of some UI elements, such as textfields, buttons, labels and so on.

For the purpose of adaptation to various screen sizes, we need to design some elements so that AUIT specified screens can enable users to use those elements to group all the UI elements on an AUIT page to form a wide variety of layouts, perform “splitting” of screens into multiple screens when all items in a form can not be sensibly displayed in one go on the display device.

We designed the system to use a “Group” element to achieve our goal. They are responsible for controlling the display of UI elements on the screen and decide if the content needs to be moved to another screen or card. For the purpose of ease to use by end users, we have designed our elements as in the column two of the following table of Figure 5.10. The new elements are designed using the principles behind “Table” elements in HTML, but with different usage.

On HTML, the script is like	On Adaptable Approach
<pre> <table> <tr> <td>...</td> <td>...</td> </tr> <tr> <td>...</td> <td>...</td> </tr> </table> </pre>	<pre> <group> <grouptr > <grouptd > content</grouptd> <grouptd > content </grouptd> </grouptr> <grouptr > <grouptd > content </grouptd> <grouptd > content </grouptd> </grouptr> </group> </pre>

Figure 5.10 Use of “group” elements

We need to fit all UI elements in the “content” place of above table. The reason is that we can have “width” or “height” values specified as properties of the “group” elements is to perform some algorithm functions within the “group/grouptr/grouptd” elements, such as splitting content to display on multiple windows. We will explain this in more detail in next section.

Based on this idea, we can create an algorithm to control page layout to display a group of elements in the one window of a device and other groups of elements in other window with links if there is not enough space to display everything in one screen. For example, the simple web page shown in Figure 5.11 contains a form used to search a list of vehicles that meet the criteria that user inputs. The whole form is composed of many UI elements and

Cell 1
 Cell 2
 Cell 3
 Cell 3
 Cell 4
 Cell 5
 Cell 6
 Cell 7
 Cell 6

Cell 4(height=20)--Label, Options, Label, Options
 would not be able to fit a mobile device screen which has only 16 characters in a line and
 maximum eight lines to display
 Screen 2
 Cell 5(height=20)--Label, Options, Label, Options
 Screen 2
 Cell 6(height=20)--Image

An approach to solve this problem is that we can specify the “minimum dividable cells”, and use an internal algorithm to divide the content of a page and display them on several screens as shown in Figure 5.11.

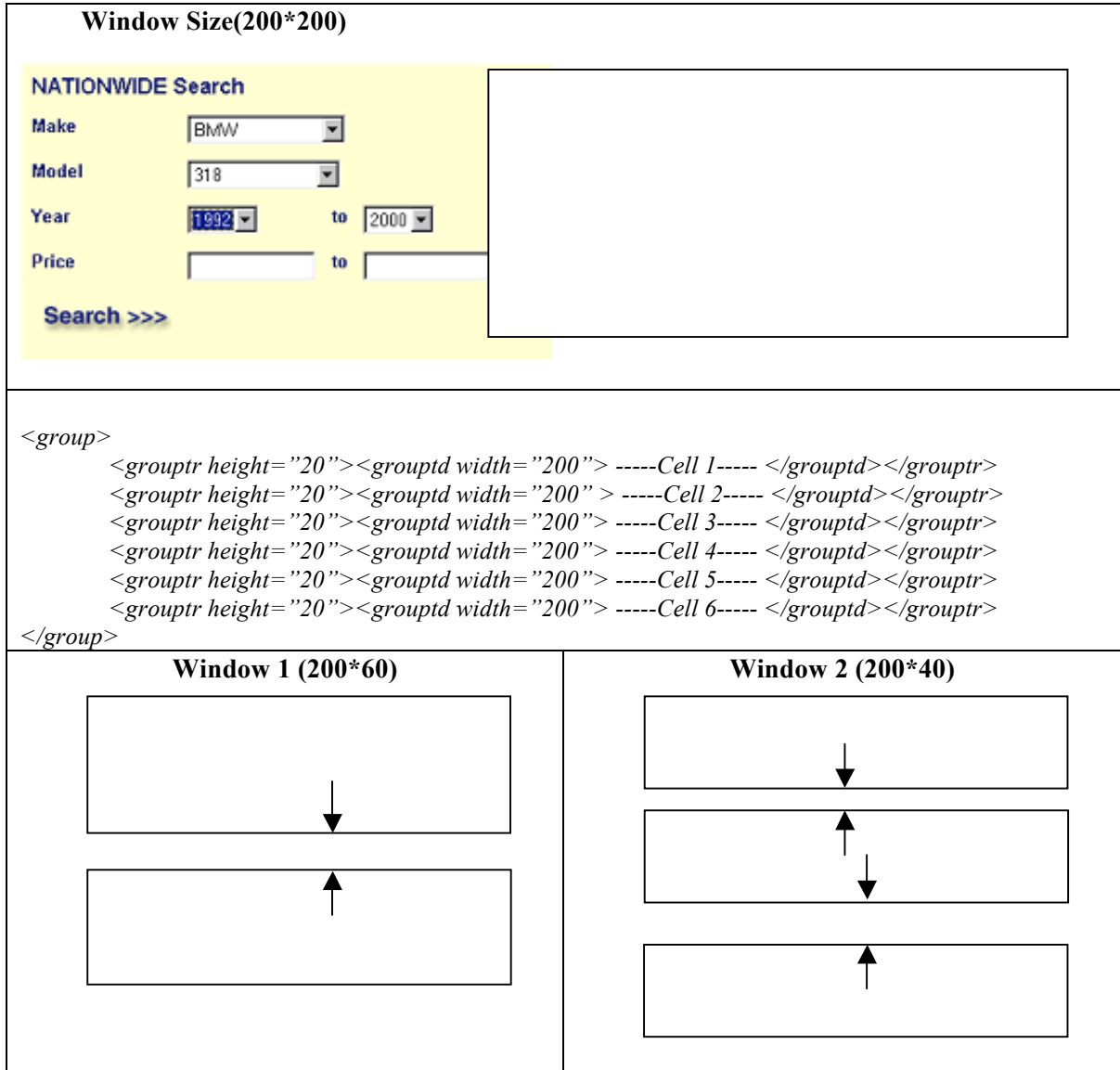


Figure 5.11 Approach to perform screen “splitting”

Figure 5.11 shows that the “Search Form” page can actually be divided into six main cells as we show on the right side, with one or more UI elements in each of them.

The purpose of specifying a minimum dividable cell is that, after we apply our algorithm, several or all “minimum dividable cells” can be grouped and placed in one screen, the others can be grouped in other screen depending on the window size of a device.

The minimum dividable cell is a group with one or more UI elements inside. It can not be divided further. The approach to specify the cell is very flexible. For example, in Figure 5.11 we should not separate “cell 2” to be two cells. “Make label” and “Make options” have to be in one screen, otherwise after performing the algorithm, these two elements could be put in two different screens which will be meaningless. So the rule to specify and group the elements is to make sure not to split elements that could make the interface meaningless. The page designer needs to specify the minimum dividable cells. In Figure 5.11, the minimum dividable cell has been specified so that each cell contains mix of text labels, textfields or dropdown lists.

For the example shown above, we have set window 1 with size of 200x60 and window 2 with size of 200x40 to view the layout of the AUIT page. In window 1 we will see Cells 1,2,3 in screen 1 and in window 2 we will see Cells 1 & 2. The arrow signs on the window are links that can link to the other pages or cells.

5.5.3.3 Algorithm

We need to apply some kind of algorithm and functions to make the system display the content properly on the device as we designed above. Our AUIT system has a database of device characteristics (screen size, colour support and default font sizes etc), that are used by our screen splitting algorithm. End users can also specify their own preferences for these different display devices characteristics, allowing for some user-specific adaptation support.

With the cell specified as shown in Figure 5.11, the algorithm is chosen to render the information showing on each page or card.

For example, we can use the above approach with rows and columns to control the layout with specified content in each cell within that table:

- Calculate the total column number and total row number for the table.
- Based on specified group column “width” and row “height” for each cell as a property of the “group/grouptr/grouptd” tag, calculate the maximum height for each row and max width for each column.
- Allocate the content for each page or card based on the specified screen size. For example, put the content of the table with total column width and total row height less than the windows width and height together, and move the rest content to next window, and so forth.

5.6 Summary

In this chapter we have introduced an adaptable approach for developing Internet applications for various devices. With some assumptions specified when doing analysis, we have created a special mark-up language which is independent of device rendering markups like HTML and WML that contains descriptions of form elements, layout and user/task relevance, unlike typical data XML encodings. We focused on analysis and design of our adaptable approach, based on the requirements that we have specified in this chapter.

In the next chapter we will use JSP technology to create a set of custom tags to implement our design ideas.

Chapter 6 - Implementation of The AUIT

6.1 Introduction

In chapter five, we have issued the system requirements, analysis and design for our Adaptable User Interface Technology (AUIT).

In this chapter we will implement AUIT (Adaptable User Interface Technology) in the form of a set of custom tags in a tag library (AUIT.tld). We have chosen JSPs as our primary implementation technology to implement our adaptable approach and to complete several aspects of adaptation, such as generation of HTML or other mark up languages and etc. We will use JavaBeans to complete the business logic.

An AUIT tag library has been designed as a package as one possible implementation option. There exist many potential paths that we can take to actually implement this approach.

6.2 JSP Custom Tags

Java Server Pages are the Java 2 Enterprise Edition (J2EE) solution for building thin-client web applications, typically used for HTML-based interfaces but also usable for building WML-based interfaces for mobile display devices.

Java Server Pages (JSP) is the mechanism for delivering dynamic Web-based content. Although JSP provides a set of predefined tags, we can also define our own tag extensions that encapsulate common functionality.

6.2.1 Overview of Tags

From experience with HTML, we already know the types of tags that can be used. There are basically two types of tags and both have different attributes.

Bodyless tag: is a tag that has a start tag but does not have a matching end tag. They are used to represent certain functions, such as presenting an input field or displaying an image.

Tags with a Body: A tag with a body has a start tag and a matching end tag. They are used to perform operations on the body content, such as formatting.

Our knowledge of tags types provides the chance to create tags based on features. The tag with a body allows us to perform content related functions between the start tag and body tag, because a set of API such as “*BodyTagSupport*” class allow us to get the body part of a tag and perform an algorithm and other functions to control the content.

6.2.2 Benefits of JSP Custom Tags

Tag libraries, or *taglibs* are a feature of JSP that enables us to build libraries of reusable JSP tags. That means user can encapsulate common behaviour in their own JSP tag and use it across the JSP pages in web applications. The ability to extract common functionality from a JSP page and easily reuse it in other pages and Web applications can be very powerful.

Typical uses of tag extensions are:

- To handle iteration over collection data structures without the need for scriptlets
- To filter or transform tag content, or even interpret it as another language.
- To introduce new scripting variables into the page.

In the following sections, we describe how to use the custom tags as the main technology to implement our approach. Custom tags look the same as the HTML tags, excepting a prefix at the beginning, so that they are easy to be used by developers who are familiar with HTML tags. The name of the custom tags should also be close to typical HTML tag names because most web designers are at least familiar with HTML.

6.2.3 Defining the Tags

Defining a custom tag is quite simple. We will describe our AUIT tags using the following steps.

- Give Name

First of all, we should give a proper name to each tag. The rule for this is that the name should be as close as possible to the related HTML or WML tag name for convenience.

- Choose Attributes.

Each tag has a set of attributes. This should follow the same rule as above. Because we will need to merge the WML and HTML, the attributes will not be exactly the same as HTML or WML, but should be close to them.

6.2.4 Build and Describe Tags

Once we have described the tags and their attributes, we can start to write the tag classes that will provide functionalities. All the custom tags in JSP are required to implement the “`javax.servlet.jsp.tagext.Tag`” Interface.

In this section, we will specify how to create a simple tag using the following example. We will create a simple tag called `<AUIT:image>` that can display images on the browser. This AUIT tag is a device independent tag and different from HTML `` and WML `` tags.

In chapter three, we have introduced the features of `` tags in HTML and WML. To implement this tag, we will create a tag called `<AUIT:image>` for displaying images on the browser.

<AUIT:image> tag has alternate text values and a range of source files (typically .gif, .jpg and wireless bit map .wbmp formats). The following example shows how to create the <AUIT:imgae> tag.

Code Example	Description
<pre>package tagext; import java.io.IOException; import javax.servlet.jsp.*; import javax.servlet.jsp.tagext.*;</pre>	<p>A tag is a Java class that implements a specialized interface. To define a simple bodyless tag, our class must implement the Tag interface. TagSupport abstract class is a convenience class that provides default implementations for all the methods in the Tag interface. Our ImageTag is to extend the TagSupport class.</p>
<pre>public class ImageTag extends TagSupport{ private String alt,src,align,writeln,device; private int height,width; private StringBuffer output=new StringBuffer(); public void setAlt(String n){this.alt=n;} public void setAlign(String n){this.align=n;} public void setSrc(String n){this.src=n;} public void setHeight(int n){this.height=n;} public void setwidth(int n){this.width=n;}</pre>	<p>Set methods can be used to handle tags that have attributes. We have set five attributes for ImageTag.</p>
<pre>public int doStartTag() throws JspException{ return SKIP_BODY; }</pre>	<p>The doStartTag method is invoked where the start tag is encountered. In this example, this method returns SKIP_BODY because this is a simple tag with no body, otherwise it should return EVAL_BODY_TAG if we need to evaluate the body of this tag</p>
<pre>public int doEndTag() throws JspTagException{ if(device=="html") output.append(" image tag syntax for HTML "); else output.append("image tag systax for WML "); return EVAL_BODY_TAG; } }</pre>	<p>The doEndTag method is invoked when the end tag is encountered. In this example, this method returns EVAL_BODY_TAG because we do not want to evaluate the rest of the page; otherwise it will return EVAL_PAGE</p>

Figure 6.1 Part of code example for a simple custom tag class and explanation

Like HTML, WML has an tag to support images. However, there are a couple of points about the tag in WML that we need to be aware of. First, the “alt” attribute of the <AUIT:image> is mandatory. Secondly, it is one of the few tags in WML that does not have a closing tag. Therefore, it needs to have its own closing mark (/) in the tag. The new “wbmp” image format stands for wireless bitmaps, which is specifically designed for wireless devices with a small monochrome screen.

For example

WML: ``

HTML: ``

Part of AUIT.tld	Description
<pre> <?xml version="1.0" encoding="ISO-8859-1"?> <!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN" "http://java.sun.com/j2ee/dtds/web- jsptaglibrary_1_1.dtd"> <taglib> <tlibversion>1.0</tlibversion> <jspversion>1.1</jspversion> <shortname>examples</shortname> <info>Simple example</info> <tag> <name>image</name> <tagclass>tagext.ImageTag</tagclass> <bodycontent>JSP</bodycontent> <info>Simple Example</info> <attribute> <name>alt</name> <required>true</required> <rtexprvalue>true</rtexprvalue> </attribute> <attribute> <name>src</name> <required>false</required> <rtexprvalue>true</rtexprvalue> </attribute> <attribute> <name>width</name> <required>false</required> <rtexprvalue>true</rtexprvalue> </attribute> </tag> </pre>	<p>The tag library descriptor, which we'll name AUIT.tld, is required to map the tag to the tag handler class and defines the way JSPs may interact with the ImageTag class. The tag library descriptor is an XML document conforming to a DTD specified by Sun Microsystems:</p> <p>The "alt" attribute of the <code><AUIT:image></code> is mandatory. So that the body for <code><required></code> tag should be true.</p>

Figure 6.2 Part of AUIT.tld and Description

This tag library can include any number of tags, so we'll add all AUIT tags to this tag library descriptor for the remaining implementation.

Code Example

```
<AUIT:image src="imagename" alt="description" width="100" height="50">
```

Because WML has different properties to HTML, we can ignore the properties that WML does not support. We need to specify the mandatory properties such as "src" and "alt" for each `<AUIT:image>` tag.

Because HTML allows either JPG or GIF graphic formats, we can write the image names the same as HTML. For WML, we need only to delete the extension of the name, and add “.wbmp” because that is the only image format that WML supports.

6.2.5 How our JSP custom tag-implemented AUIT page work

A developer writes an AUIT-encoded screen specification which makes use of JavaBeans (basically Java classes) to process form input data and to access Enterprise JavaBeans, databases and legacy systems. At run-time the AUIT tags are processed by the JSPs using the custom tag library classes we have written. When the JSP encounters an AUIT tag, it looks for a corresponding custom tag library class that it invokes with tag properties. This custom tag class performs suitable adaptations and generates appropriate output text to be sent to the user’s display device. Link and submit tags produce HTML or WML markups directing the display device to other pages or to perform a POST of input data to the web server as appropriate. Figure 6.3 outlines the way processing of AUIT tags is done. Note that dynamic content scriptlet code can be interspersed with AUIT tags (not illustrated here).

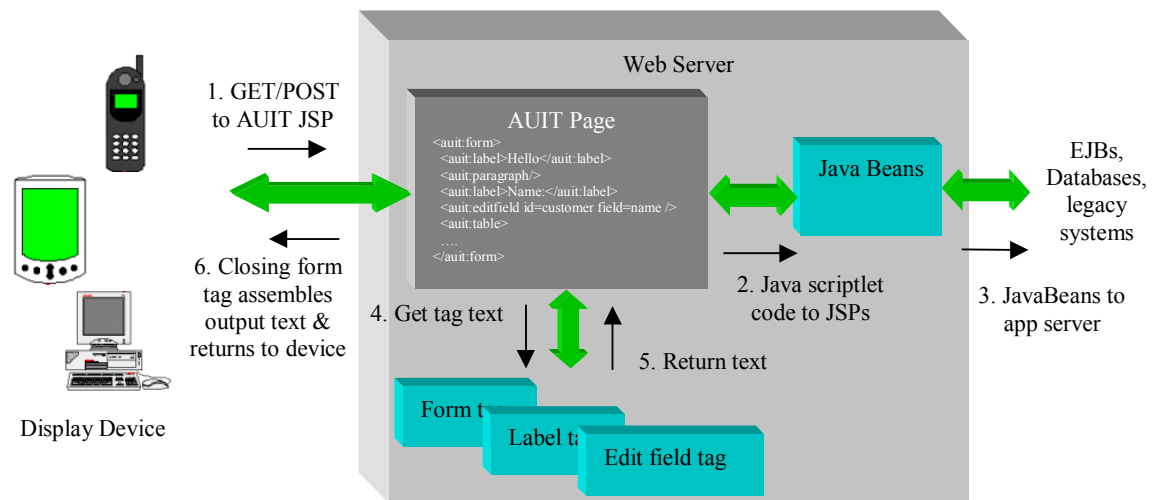


Figure 6.3 How our JSP custom tag-implemented AUIT pages work

6.3 Implementation of AUIT tags

In section 6.2, we introduced JSP custom tags to implement our designs. The objective is to develop a set of device-independent screen element tags that allow developers to specify their screens, independent of user, task and display device.

We have outlined some of these tags in Figure 6.4 along with typical mappings to HTML and WML mark-up tags.

Screen Element Tag	Description	HTML	WML
<AUIT:template>	Encloses contents of whole screen	<html>	<wml>
<AUIT:form>	Indicates an input form to process	<form>	<do type=accept>
<AUIT:group> <AUIT:grouptr> <AUIT:grouptd>	Used for group one or more UI elements in screen, and performs screen “splitting” for adaptation of various screen sizes.	-	-
<AUIT:table> <AUIT:tr> <AUIT:td>	Displays UI elements in a group cell in a table format.	<table>	<table>
<AUIT:iterator>	Iterates over data structure elements	-	-
<AUIT:break>	Line break	 	
<AUIT:label>	Label on form	Plain text	Plain text
<AUIT:layout>	Describe the display format for label		
<AUIT:textfield>	Edit field description	<input type=text>	<input type=text>
<AUIT:radio>	Radio button	<input type=radio>	<input type=radio>
<AUIT:select>	Popup menu item list	<select ...>	<select ...>
<AUIT:image>	Image placeholder		
<AUIT:link>	Hypertext link		<go href=...>
<auit:submit>	Submit button (for POST)	<input type=submit>	<do><go href=...>
<% ... %>	Embedded Java scriptlet code	-	-

Figure 6.4 Examples of AUIT tags and corresponding HTML,WML tags.

Each kind of AUIT tag has various properties the developer can specify, some mandatory and some optional. Tags generally either layout control (form, group, table, row, paragraph etc), page content (edit field, label, line, image etc), or control inter-page navigation (submit, link).

Therefore, our task becomes to create a set of custom tags to make up our AUIT page. Our AUIT pages are actually Java Server Pages (JSPs) that contain a number of JSP custom tags. In the following sections, we will describe some of the AUIT custom tags in further detail.

6.3.1 Build Document Structure Tags -Template Tag

The Template tag is a most important tag. It appears on the top of each AUIT page and performs three main functions, which are:

- Identifying the device
- Providing AUIT page structure
- Output the layout at close tag of `</AUIT:template>`.

6.3.1.1 Identifying the Client's Capabilities to Serve Appropriate Content

In section 4.4.3.2 of chapter four, we explained how we ascertain user device information. In the traditional approach, because we have different version of “JSPs” for different devices, we only need to detect the device information once in the “index” page. But using AUIT, we have one AUIT page for all devices, so we need to detect the device information in each AUIT page. Because we need to use that device information in almost all AUIT tags within an AUIT page, we need to detect the device information at the very beginning of each AUIT page and store the data as page variable that can be used by any other AUIT tags.

The content dynamically generated by an AUIT page must be in a format supported by the requesting device. The identification of the capabilities of a requesting client means, in our case, specifically identifying the proper format in which to respond. For example, if a mobile device requests service, we need to provide data in WML format.

Again we need to use the same approach to get device information as we described in chapter four. Based on the type of user agent from the header information, we have a “page variable” to store the device information so that it can be fetched by any AUIT tags. However, we have put all these functions in a “templateTag” class, instead of an “index” page as specified in Chapter four.

Almost all the custom tags in the AUIT page need the device information, so that after identify the device type, the device information can be stored as an attribute using `setAttribute()` method. It can be accessed and retrieved by all of the elements using `getAttribute()` method.

Code Example:

```
pageContext.getRequest().setAttribute("device", userAgent)
```

6.3.1.2 Specify AUIT Document Structure

Another function that `<AUIT:template>` provides is to describe our AUIT page structure. From the design for the basic structure of an AUIT page specified in Chapter five, each AUIT page implemented by JSP custom tag should have the following basic page structure.

```
<AUIT:template>
  <AUIT:group>
    <AUIT:grouptr height="...">
      <AUIT:grouptd width="...">
        ...
      </AUIT:grouptd>
      <AUIT:grouptd width="...">
        ...
      </AUIT:grouptd>
    </AUIT:grouptr>
  </AUIT:group>
</AUIT:template>
```

Figure 6.5 AUIT page structure

The code shown in Figure 6.6 is part of the “`templateTag`” class. This generates the WML or HTML document structure respectively appropriate for the requesting device detected previously.

```
if(device=="html")
    output.append("<html><head><title>"+name+"</title></head><body>")
else{
    output.append("<?xml version='1.0'?>");
    pageContext.getResponse().setContentType("text/vnd.wap.wml");
    output.append("<!DOCTYPE wml PUBLIC '-//WAPFORUM//DTD WML 1.1//EN'
        'http://www.wapforum.org/DTD/wml_1.1.xml'><wml>");
}
```

Figure 6.6 Part of code in “`templateTag`” class

6.3.1.3 Output the layout

To output the layout we need to create a “StringBuffer” object called “output” in “doStartTag” method of “templateTag” class and have assigned it as a value of “output” attribute. This “output” variable can be retrieved by any custom tags specified between <AUIT:templage> and </AUIT:template>.

Figure 6.7 shows the code example of “templateTag” class that processes the “output”.

```
public int doStartTag() throws JspTagException{
    StringBuffer output=new StringBuffer();
    pageContext.getRequest().setAttribute("output",output);
    ...
    return EVAL_BODY_TAG;
}
public int doEndTag() throws JspTagException{
    output=(StringBuffer)pageContext.getRequest().getAttribute("output");
    ...
    bodyContent.getEnclosingWriter().write(output.toString());
}
}
```

Figure 6.7 Part of code in templateTag class

There is a difference between our approach and a normal JSP page that contains several custom tags. Our approach will write out the “StringBuffer” output at the very end of the JSP page in “doEndTag” method of “templateClass” and send it to the requesting client by server. The output needs to be in the language format that the device supports. But a normal JSP page will usually write output at the end of each custom tag.

The benefit of our approach is that we can pass the content in StringBuffer - “output” for further processing, such as dropping off or adding content at any stage.

For example lets say we need to pass all the elements enclosed in “group” elements into a function bean for “splitting screen” using the algorithm as discussed previously. The algorithm will decide if there is need to split the screen and what to display on the

first screen. When the end tag of the template tag is reached, we will write output to the first screen, which might be just one quarter of the entire content of the AUIT page.

6.3.1.4 Sequence Diagram and Description

The sequence diagram shown in Figure 6.8 can help us to understand the whole process.

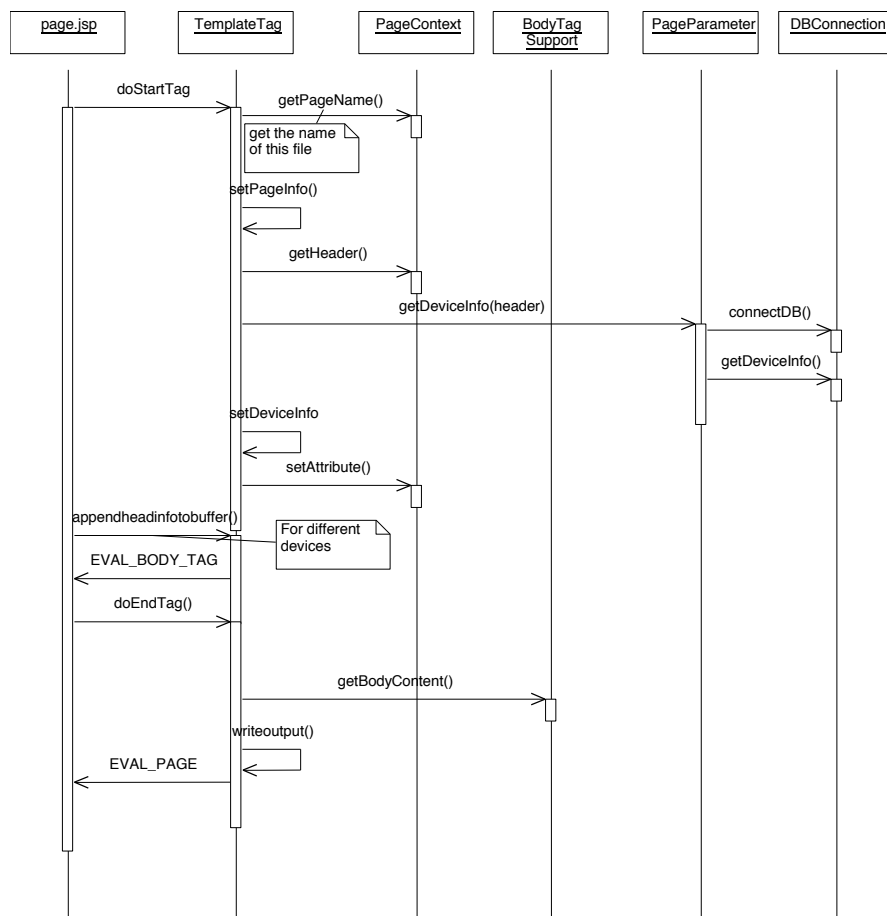


Figure 6.8 Sequence Diagram of “templateTag” class

6.3.2 Build Page Flow Control Elements

From the design that we introduced in chapter five we have seen the description of `<group>`, `<grouptr>` and `<grouptd>` tags. Now we need to implement these three tags using JSP custom tags.

6.3.2.1 Using “Group/Grouptr/Grouptd” to Control Layout

<AUIT:group> ,<AUIT:grouptr> and <AUIT:grouptr> tags are used when auto-laying out AUIT elements enclosed in group cells. These tags are designed for completing several adaptation tasks.

One or more AUIT UI elements can be included in the <AUIT:grouptd> as cells just as “table/tr/tr” tags are used in HTML. Each of these AUIT tags has various properties which are listed in the following table.

Tag	Properties
<AUIT:group>	width (mandatory): indicates the entire width of the device screen weight (mandatory): indicates the entire height of the device screen align (optional) : indicates the content alignment among the device screen(for html) bgcolor (optional) : indicates the background colour (for html)
<AUIT:grouptr>	Height(optional) : indicates the width pixels of the cell takes
<AUIT:grouptd>	width : indicates the height of the cell takes colspan : shows the number of columns the cell will take. align : the cell alignment

Figure 6.9 “Group/grouptr/grouptd” tags and their properties

In chapter five we introduced the most important functions that these elements provide; being to control page layout and perform “splitting” of screens into multiple screens when all items in a form cannot be sensibly displayed in one go. We have also introduced the algorithm needed to complete these tasks.

For example, the following screen dump was developed using AUIT tags and rendered by an IE5.0 browser on standard desktop window with size 800x600 pixels.

Search Result					
Status	Year	Vehicle	Kms	Price	Condition
For Sale	1996	Honda Camry	34500	\$6500	Good
For Sale	1994	Honda Celica	68000	\$6300	Excellent
Sold	1994	BMW Celica	67000	\$8000	Good
For Sale	1995	Toyota Camry	10000	\$5800	Good
Sold	1992	Toyota Celica	71000	\$6200	Good
For Sale	1993	Toyota Corolla	81900	\$7800	Excellent

Figure 6.10 Search Result Interface displayed on a standard browser

This is a JSP page comprised of AUIT tags. Some of the tags we will introduce later; for now we will focus on how `<AUIT:group>`, `<AUIT:grouptr>` and `<AUIT:grouptd>` control the screen splitting. All the transformations will be done automatically by backend code. Figure 6.11 shows the code for the page in Figure 6.10.

```

<%@ taglib uri="/AUIT" prefix="AUIT" %>
<jsp:useBean id="testBeanId" scope="page" class="beans.FunctionBean" />
<%testBeanId.connectDB();%>
<AUIT:template>

  <AUIT:group width="800" height="800" >

    <AUIT:grouptr cellheight="100" >
      <AUIT:grouptd cellwidth="100" colspan="5">
        <AUIT:layout size="+3" bold="b" color="#101077">
          <AUIT:label text="Search Result"></AUIT:label>
        </AUIT:layout>
      </AUIT:grouptd>
    </AUIT:grouptr>
    <AUIT:grouptr cellheight="100" >
      <AUIT:grouptd cellwidth="100">
        <AUIT:layout size="3" color="#101077" bold="b">
          <AUIT:label text="Status"></AUIT:label></AUIT:layout>
        </AUIT:grouptd>
      <AUIT:grouptd cellwidth="100">
        <AUIT:layout size="3" color="#101077" bold="b">
          <AUIT:label text="Year"></AUIT:label></AUIT:layout>
        </AUIT:grouptd>
      <AUIT:grouptd cellwidth="100">
        <AUIT:layout size="3" color="#101077" bold="b">
          <AUIT:label text="Vehicle"></AUIT:label></AUIT:layout>
        </AUIT:grouptd>
      <AUIT:grouptd cellwidth="100">
        <AUIT:layout size="3" color="#101077" bold="b">
          <AUIT:label text="Kms"></AUIT:label></AUIT:layout>
        </AUIT:grouptd>
      <AUIT:grouptd cellwidth="100">
        <AUIT:layout size="3" color="#101077" bold="b">
          <AUIT:label text="Price" ></AUIT:label></AUIT:layout>
        </AUIT:grouptd>
      <AUIT:grouptd cellwidth="100">
        <AUIT:layout size="3" color="#101077" bold="b">
          <AUIT:label text="Condition"></AUIT:label></AUIT:layout>
        </AUIT:grouptd>
    </AUIT:grouptr>
    <% java.util.List carlist=(java.util.List)testBeanId.getCarList();%>
    <%java.util.Hashtable car=(java.util.Hashtable)carlist.get(index.intValue());%>

    <AUIT:iterator collection="<%=carlist%>" >

```



```

2
<AUIT:grouptr cellheight="100" >
  <AUIT:grouptd cellwidth="100">
    <AUIT:label text='<%= (String) car.get ("status") %>'></AUIT:label>
  </AUIT:grouptd>
  <AUIT:grouptd cellwidth="100">
    <AUIT:label text='<%= (String) car.get ("year") %>'></AUIT:label>
  </AUIT:grouptd>
  <AUIT:grouptd cellwidth="100">
    <AUIT:label text='<%= (String) car.get ("vehicle") %>'></AUIT:label>
  </AUIT:grouptd>
  <AUIT:grouptd cellwidth="100">
    <AUIT:label text='<%= (String) car.get ("kilos") %>'></AUIT:label>
  </AUIT:grouptd>
  <AUIT:grouptd cellwidth="100">
    <AUIT:label text='<%= "$"+ (String) car.get ("price") %>'></AUIT:label>
  </AUIT:grouptd>
  <AUIT:grouptd cellwidth="100">
    <AUIT:label text='<%= (String) car.get ("condition") %>'></AUIT:label>
  </AUIT:grouptd>
</AUIT:grouptr>

</AUIT:iterator>

</AUIT:group>

</AUIT:template>

```

Figure 6.11 Code Example for “Search Result” page

Apart from the general desktop browser with a standard window size of 800x600, we have used another two devices to browse this page which are shown in Figure 6.10.

Window 1: 160x160

Window 2: 16characters*5lines

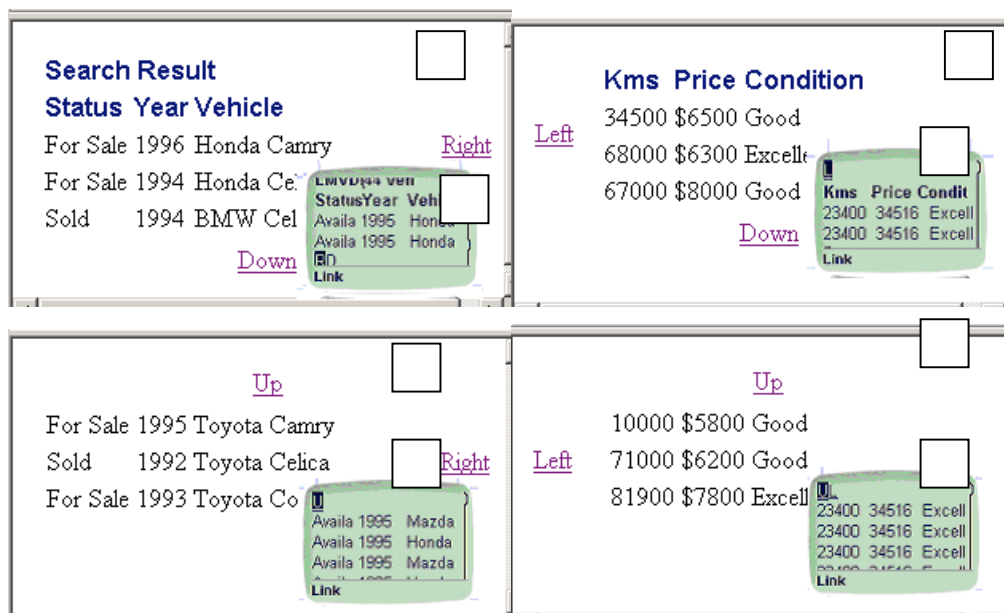


Figure 6.12 Page dumps for screen splitting performed on two devices

The “Down”, “Right”, “Up” and “Left” links shown on window 1 of Figure 6.12 are generated automatically. In window 2, we see “D”, “R”, “U” and “L” links which fulfil the same purpose. The content that can be displayed in one screen of a desktop browser has been spit into four.

Saved information is stored in four “StringBuffer”s instead of one and only content in the first “StringBuffer” has been output. To view the data on the other three pages the user can click on the links.

The links are generated using this approach shown in Figure 6.13 for HTML and WML enabled files.

For devices support HTML	For devices support WML																
<pre><html> <body> <table border="1" style="width: 100%; text-align: center;"> <tr> <td style="width: 50%;">Right (link to10)</td> <td style="width: 50%;">Left (link to 00)</td> </tr> <tr> <td>Screen 00</td> <td>Screen 01</td> </tr> <tr> <td>Down (link to 01)</td> <td>Down (link to 11)</td> </tr> <tr> <td>Up (link to 00)</td> <td>Up (link to 01)</td> </tr> <tr> <td>Screen 10</td> <td>Screen 11</td> </tr> <tr> <td>Right (link=11)</td> <td>Left (link to10)</td> </tr> </table> </body> </html></pre>	Right (link to10)	Left (link to 00)	Screen 00	Screen 01	Down (link to 01)	Down (link to 11)	Up (link to 00)	Up (link to 01)	Screen 10	Screen 11	Right (link=11)	Left (link to10)	<pre><WML> <table border="1" style="width: 100%;"> <tr> <td style="width: 50%; vertical-align: top;"> <pre><Card id="00"> R(link to10) Card 00 D(link to10) </Card></pre> </td> <td style="width: 50%; vertical-align: top;"> <pre><Card id="01"> L(link to 00) Card 01 D(link to 11) </Card></pre> </td> </tr> <tr> <td style="vertical-align: top;"> <pre><Card id="10"> U(link to 00) Card 10 R(link to11) </Card></pre> </td> <td style="vertical-align: top;"> <pre><Card id="11"> U(link to 01) Card 11 L(link to 10) </Card></pre> </td> </tr> </table> </WML></pre>	<pre><Card id="00"> R(link to10) Card 00 D(link to10) </Card></pre>	<pre><Card id="01"> L(link to 00) Card 01 D(link to 11) </Card></pre>	<pre><Card id="10"> U(link to 00) Card 10 R(link to11) </Card></pre>	<pre><Card id="11"> U(link to 01) Card 11 L(link to 10) </Card></pre>
Right (link to10)	Left (link to 00)																
Screen 00	Screen 01																
Down (link to 01)	Down (link to 11)																
Up (link to 00)	Up (link to 01)																
Screen 10	Screen 11																
Right (link=11)	Left (link to10)																
<pre><Card id="00"> R(link to10) Card 00 D(link to10) </Card></pre>	<pre><Card id="01"> L(link to 00) Card 01 D(link to 11) </Card></pre>																
<pre><Card id="10"> U(link to 00) Card 10 R(link to11) </Card></pre>	<pre><Card id="11"> U(link to 01) Card 11 L(link to 10) </Card></pre>																

Figure 6.13 Example to show links for various devices

Figure 6.13 shows the generated files and the page content links of a “search results” page which has been split into four parts. To generate relevant links for HTML and WML enabled devices, we need to process them separately in terms the device’s specific requirements for this page.

When a WML enabled mobile device receives this deck (jsp file), it loads the first card automatically, and the user can navigate using the soft key on the device.

6.3.2.2 Showing the Table in a Card.

There is limitation on “table” tag in WML markup language that are different to the HTML equivalents. *Wml will only allow text and image type elements within the cell.* Getting the same table layout as shown in Window 1 of Figure 6.12 using wml is not as easy as in HTML.

One approach is to specify the alignment for each cell manually, deleting non-essential characters. To implement this approach, we need to count the number of characters that need to be displayed on each cell.

For example, lets say we allocate “100” pixels as “width” for a cell on which we need to display a line of characters. In the database, we have pre-stored some data such as the number of pixels that a character will take for each device. If a character will take 5 pixels on a PDA, the number of characters that our device can display per line will be $100/5=20$. This example is shown in Figure 6.14.



Status/Year	Vehicle
Avails 1995	Honda
Avails 1995	Honda

Link

Figure 6.14 Page dump for display table in a card

With the above value, we can calculate the maximum number of the characters that each cell could contain. We will then delete the rest of characters and only display the first five characters for each cell.

Due to the full automation cutting of the characters, the limitation with this approach is that some characters with important information might be cut as well, such as price of a car.

6.3.3 Build UI Control Elements

UI control elements are a set of elements to control the layout of UI elements. They will not affect the layout of each UI element itself, but the way they display on the screen.

They decide if a group of UI elements will be displayed and/or the way that they will appear in the window.

These elements include:

1. User/Task/Device controlling which elements will be shown
2. Table, tr and td will display a group of elements in table format
3. Iterator will iterate a group of elements on the window

6.3.3.1 Build User/Task/Device Content Control Tags

As we have shown in previous chapter, a page can show or hide the content based on the particular user, task and device value. In a traditional approach, we used the “if/else” statement to switch off part of the content of a page to be displayed. In AUIT, we have design three elements <AUIT:user>, <AUIT:task>, <AUIT:device> to perform these tasks.

- **Use of these AUIT tags**

The following are examples of how we use these tags and how they work. These are demonstrated on pages we have developed in our car site, using AUIT.

Example One: Use <AUIT:user> tag

This code is part of the AUIT page for “search results”, being a list of cars. From the login page, we have created a “cookie” so that we can identify the user. For this example, if the user is a registered dealer, page (1) in Figure 6.15 will show an extra

column with an "Update" link on it that allows the dealer to update their information. Otherwise page (2) will show.

```

<AUIT:user user="dealer" value="dealer">
  <AUIT:grouptd cellwidth="100">
    <AUIT:link direct="" param="false">
      <AUIT:label text='Update' allowcut='false'></AUIT:label>
    </AUIT:link>
  </AUIT:grouptd>
</AUIT:user>

```

Figure 6.15 Code example for " user" adaptation

Example Two: Use <AUIT:task> tag

The two pages shown in Figure 6.16 are from same AUIT page called "searchresult.jsp". The first one is shown when a user searches a list of cars and the second page is shown when a user searches a list of dealers. The different layout is generated from the different task value passed into this page. The task value has to be passed and identified on this page.

Figure 6.16 " searchresult.jsp"

The code shown in Figure 6.17 is the part of the code for showing the above two page dumps.

<AUIT:grouptr cellheight="100" >	
<pre> <AUIT:task task="task" value="car"> <% java.util.List dealerfields=(java.util.List) testBeanId.g etCarFields();%> <AUIT:iterator collection="<%=carfields%>"> <AUIT:grouptd cellwidth="100"> <AUIT:label text=" <%= (String) carfields.get (index.intValue ())%>"> </AUIT:label> </AUIT:layout> </AUIT:grouptd> </AUIT:iterator> </AUIT:task> </pre>	<pre> <AUIT:task task="task" value="dealer"> <% java.util.List dealerfields=(java.util.List) testBeanId.ge tDealerFields();%> <AUIT:iterator collection="<%=dealerfields%>"> <AUIT:grouptd cellwidth="100"> <AUIT:label text=" <%= (String) dealerfields.get (index.intValue ())%>"> </AUIT:label> </AUIT:layout> </AUIT:grouptd> </AUIT:iterator> </AUIT:task> </pre>
</AUIT:grouptr>	

Figure 6.17 Code example for using <AUIT:task>

Example Three: Use <AUIT:device> tag

The two pages shown in Figure 6.18 are from same AUIT page called “HomePage.jsp” but it displays different content on different devices. The first page shows three featured cars when the user is viewing the page using a browser on a desktop PC. The second page would not show on a mobile device with small window size.

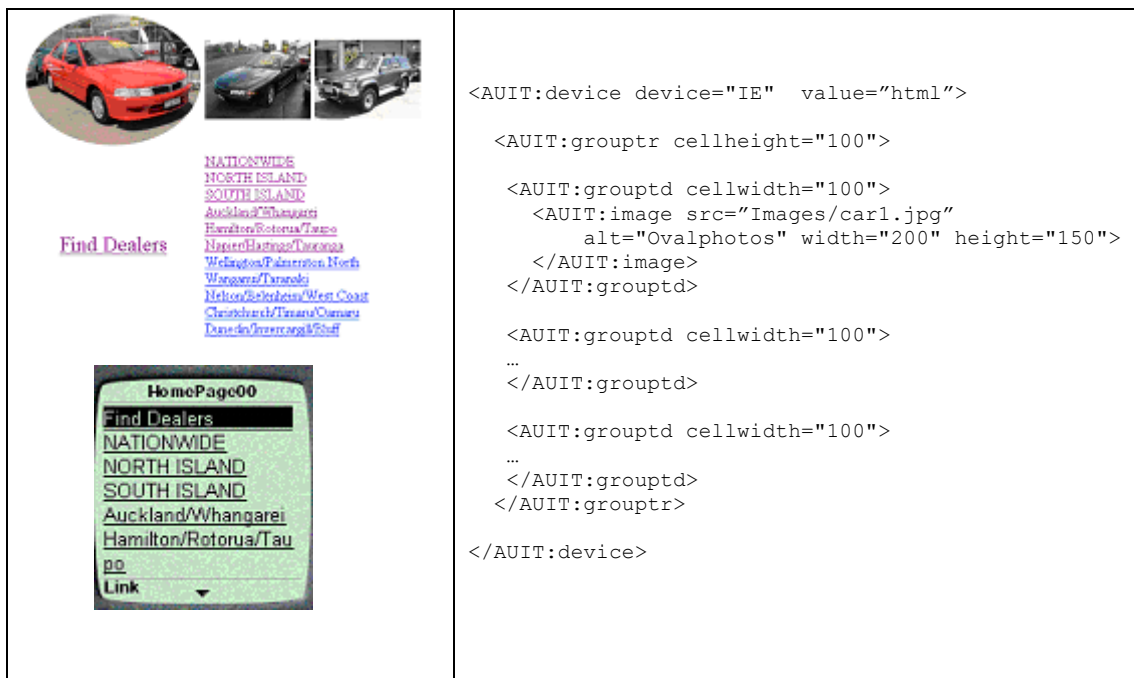


Figure 6.18 Page dump and Code example for “device” adaptation

- **How the tags work**

Whenever we use these UI control tags, we need to get the proper value of user, task and device in a page as a parameter or attribute. The device identifying information is detected in <AUIT:template> through the header information .

The user information is stored in a cookie. After a cookie is defined successfully, the value of the cookie can be fetched anywhere on a web site. We stored our user information in a cookie after a user registered and logged in for the first time. For example, in Figure 6.15, the “update” link is shown only when the detected user value is “dealer”, not for others. For a page to detect the user information, the user needs to login and we will save the user value throughout the cookie until that user logs off. All the content displayed will be user-specific. Obtaining & storing the “user” values is easily achieved provided that the users system supports cookies. Of course, some of the devices just don’t support cookies, so the user information cannot be received or set properly. However, for the purpose of this exercise we have assumed that all devices we are writing for do support cookies, so we are not going to focus on solving this problem in this thesis.

Task information is usually acquired as it is passed between pages by a variable either via a “form” or a “link”. The data can be captured through either a form or a link using the “getParameter()” method that is provided by JSP.

After we have received these values, we can compare them with the one specified as an attribute in the AUIT tag.

For example, if we have the value “car” as a task variable from the above method, the following code shows the “content” in between the start tag and end tag will be displayed.

<p>Code Example for how to use “task” in AUIT page</p> <pre><AUIT:task task="task" value="car"> ...content.. </AUIT></pre>
<p>Code Example of “TaskTag”</p> <pre>public int doStartTag() throws JspTagException{ supertask=(String)pageContext.getRequest().getParameter("task"); if(!supertask.equals(value)){ return SKIP_BODY; } return EVAL_BODY_TAG; }</pre>

Figure 6.19 Explanation of the task based adaptation

6.3.3.2 Table Elements

Figure 6.20 shows how to use the “table/tr/td” AUIT tags to create the aligned UI layout.



 <p style="text-align: center;">HTML</p>  <p style="text-align: center;">WML</p>	<pre> <AUIT:table> <AUIT:tr> <AUIT:td> <AUIT:label text="Login"></AUIT:label> </AUIT:td> </AUIT:tr> <AUIT:tr> <AUIT:tr> <AUIT:td> <AUIT:label text="User Name:"></AUIT:label> </AUIT:td> <AUIT:td> <AUIT:textfield type="text" name="login" size="12" value="wendy"> </AUIT:textfield> </AUIT:td> </AUIT:tr> <AUIT:tr> <AUIT:tr> <AUIT:td> <AUIT:layout color="green"> <AUIT:label text="Password:"></AUIT:label> </AUIT:layout> </AUIT:td> <AUIT:td> <AUIT:textfield type="password" name="pw" value=""> </AUIT:textfield> </AUIT:td> </AUIT:tr> <AUIT:tr> <AUIT:td> <AUIT:button name="submit" type="submit" value="Submit"> </AUIT:button> </AUIT:td> </AUIT:tr> </AUIT:table> </pre>
--	--

Figure 6.20 Page dump and Code example

Both WML and HTML provide “table” elements which combine with the “tr” and “td” elements to create sets of aligned UI layout. The “table” elements determine the structure of the columns and rows.

WML will only allow the “text” and “images” to be displayed within the cells of a table. In AUIT, we have used a generalized way to solve this problem. That is to leave the “table” elements and its attributes when adapting the device to display HTML. Then check the type of element displayed in each cell when the device uses WML. If any element type is neither text, nor image then AUIT will switch to use a <AUIT:break> tag when starting a new row rather than using “table/tr/td” tags.

6.3.3.3 Building Other Functional Tags –“Iteration”

Iteration is an example for handling some of the functions using custom tags. In a web page, we often need to iterate data as we can see in Figure 6.16.

- *AUIT*: Define a custom tag that will handle the iteration, as in the following example. The tag handler will then perform the processing logic with each iteration, before processing its body content.

```
<AUIT::iteration collection="<%=dealerlist%>">
    ...
</AUIT:iteration>
```

- *Traditional*: It is acceptable to use scriptlets to do this. Embed some of the Java code in the content of the pages.

```
<%
    int num=dealerlist.size();
    for(int i=0;i< num;;i++){
%>
...
<%}%>
```

Compare the AUIT and traditional approach examples. The AUIT one is obviously neat and more readable. Control flow is handled more cleanly.

6.3.4 Building UI Elements

In chapter five, we have designed a set of UI elements. Now we will explain how we implement them.

6.3.4.1 Text and Layout Formatting Elements

The `<AUIT:label>` and `<AUIT:layout>` tag are two of the fairly simple tags.

`<AUIT:label>` can display normal text on the device, and `<AUIT:layout>` can be used to format the text, eg: such as set the size or colour.

Because some devices such as the mobile phone cannot display full text we can pass the text of the `<AUIT:label>` element to the server side for processing. This is the main reason we design a tag to have the text information specified as the text attribute of the `<AUIT:label>` instead of writing them as text straightforward in the AUIT page.

If we have a long text string that cannot fit on a small screen, we can use the following approaches. We provide three options to handle the text-display problem on different screens.

- Use **<AUIT:device>** tag

Display different text for different devices as shown on the following code.

```
<AUIT:device name="IE" value="html">
    <AUIT:label text="Search Result"/>
</AUIT:device>
<AUIT:device name="Nokia" value="wml">
    <AUIT:label text="Result"/>
</AUIT:device>
```

Figure 6.21 Use **<AUIT:device>** to control text display

- Use **“allowcut”** attribute.

If there is no content control element specified, the text displayed can be also processed. Long text can be cut out automatically to fit into the window.

```
<AUIT:layout face="Arial, Helvetica, sans-serif" size="+3" bold="b" color="#101077">
    <AUIT:label text="Search Result" allowcut="true"></AUIT:label>
</AUIT:layout>
```

Figure 6.22 Use **“allowcut”** to control text display

For displaying a table on a mobile phone as we described in section 6.3.2.2, we need to cut off some part of a word in order to fit it. But some of the words should not be cut without altering the meaning, such as the odometer reading or price of a car, so we need to have some attribute in the code to specify what “not to cut”. If we look at the above code carefully, we can see that there is an attribute in the “label” tag called “allowcut” to specify this function. The value “true” of this element means that part of the text can be cut as necessary.

```

<AUIT:form action='login' method='post' name='form'>
  <AUIT:grouptr cellheight="..">
    <AUIT:grouptd cellwidth="..">
      < AUIT:label text="User Name:"></AUIT:label>
      < AUIT:textfield type="text" name="login"> </AUIT:textfield>
    <AUIT:label text="Password:"></wendy:label>
    < AUIT:textfield type="password" name="password"></AUIT:textfield>
    <AUIT:button name="submit" type="submit" value="Submit"></AUIT:button>
  </AUIT:grouptr>
</AUIT:form>

```

6.3.4.2 Building Form and User Input Elements

The form and user input elements provide a mechanism for capturing user input. Unlike the user input paradigm in HTML where a user enters content in fields and finally clicks a Submit button to send the input off to the server, WML does not have a specific Submit button input element. Rather, WML input is sent to the server when the user selects the soft-key label associated with the <do> element.

- **Elements in the form.**

Form tags in AUIT pages let the designer write code just as they can for HTML. More than that, form tags are designed on top of an object-oriented programming model, enabling code reuse and separation of the application code from page content. We can draw the controls on a form, then implement the event procedures underneath.

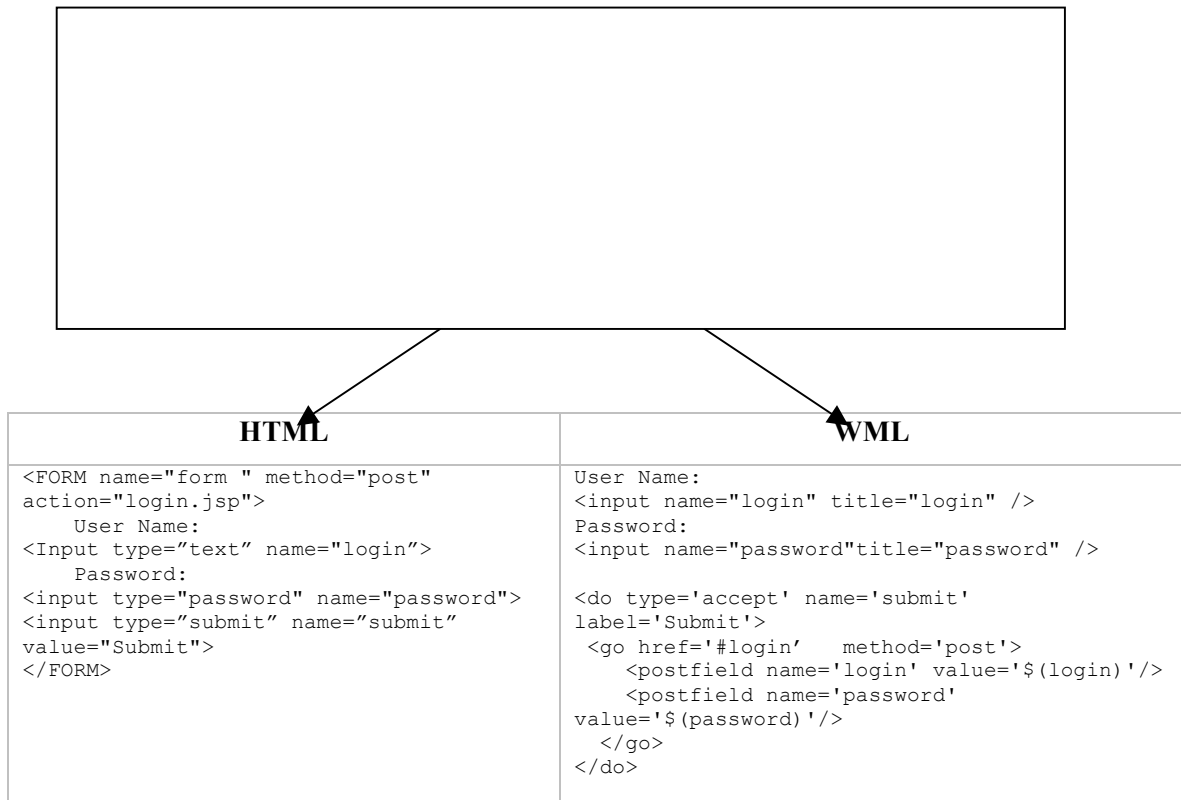


Figure 6.23 Code example using form elements

The form of other input elements are basically the same as HTML; even simpler. We can find them from the diagram shown in Figure 6.23.

The code on the left generated is HTML. It performs a standard form post that sends the user input back to the same file. The second is WML, where a WML enabled device is used.

Other form input elements developed include “select/option”, button, etc. Note one point, both HTML and WML allow “select” and “options” and they have similar syntax. Normally “options” are generated from the database. For example, in the car site we allow people to choose the make and model. In HTML we normally provide a “select/option” dropdown list allowing hundreds of choices, but for other mobile device, because of the memory and screen size limitations, we could not display that many choices. One way to solve this problem is to write the following code.

```
<AUIT:device device="html" >
  < AUIT:select name="to">
    < AUIT:option value="Wendy"></ AUIT:option>
    < AUIT:option value="James"></ AUIT:option>
    < AUIT:option value="Ming"></ AUIT:option>
  </ AUIT:select>
</AUIT:device>
<AUIT:device device="wml">
  < AUIT:textfield name="to" type="text" size="8"></ AUIT:textfield>
</AUIT:device>
```

Figure 6.24 Code example to display “option/select” elements

This replaces the drop-down menu with a text input field.

- **Structure to specify a form**

To specify a form we should use the structure specified in Figure 6.25. <AUIT:form> tag is used between the <AUIT:group> and <AUIT:grouptr>.

```
<AUIT:group>
  <AUIT:form ...>
    <AUIT:grouptr> <AUIT:grouptd>
    <AUIT:grouptr> <AUIT:grouptd>
  </AUIT:form>
  <AUIT:grouptr> <AUIT:grouptd>
</AUIT:group>
```

Figure 6.25 Code Example to specify <AUIT:form>

6.3.4.3 Build Navigation Elements

WML uses anchors (<anchor>) which are the WML counterpart of <A> tags in HTML. An anchor can be defined as:

Code example

```
<anchor>follow me
  <go href="destination">
</anchor>
```

is identical semantically to the following markup

```
<a href="destination">follow me</a>
```

We have created an <AUIT:link> tag to allow the user to specify pages to go to and actions for the target JSP to perform. But there are also some system-generated links we have introduced previously, such as the “right” link.

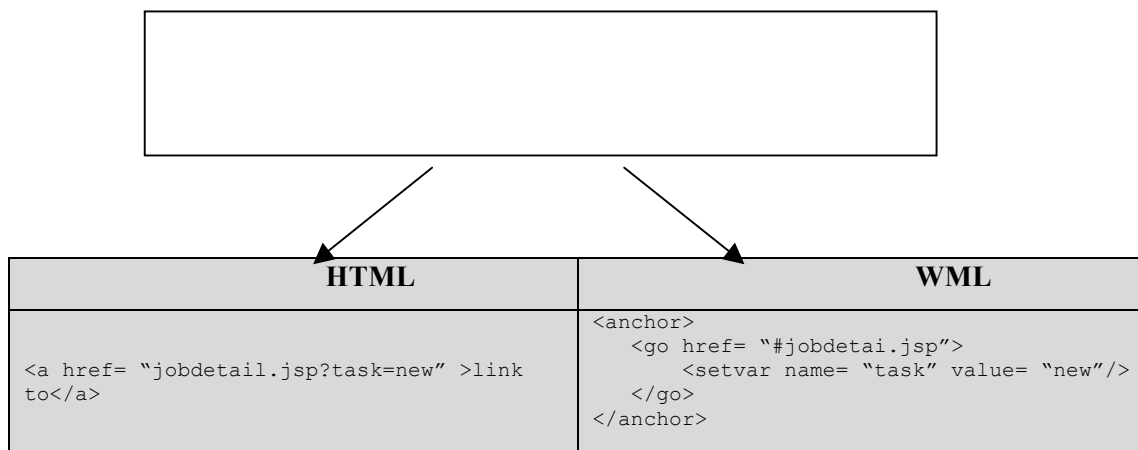


Figure 6.26 Code Example of Navigation Elements

6.3.5 System Deployment

Now that we have defined our AUIT tags and written their supporting class, we now need to tell the web application where to find the tags by registering the “taglib” with it. The Web application configuration file (web.xml) is stored under the “web-inf” directory. Open up the “web.xml” file and insert the following XML fragment.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <display-name>tagext</display-name>
  <description>Tag extensions</description>
  <session-config>
    <session-timeout>0</session-timeout>
  </session-config>
  <taglib>
    <taglib-uri>
      /AUIT
    </taglib-uri>
    <taglib-location>
      /WEB-INF/tlds/AUIT.tld
    </taglib-location>
  </taglib>
</web-app>
```

Figure 6.27 Code in “web.xml” file

6.4 Building Interfaces to Make the System Configurable

In this section, I will introduce some methods for developers to use in order to store the system-required data. Regardless of the application related data, like “car” and “dealer” information in the car site, the device and user information also needs to be considered. For example, what type of client device and user will use this application. User preferences as well as device type may vary. For example, user may want to use a non-standard window size to view the application. Extensibility needs to be built in to allow new user preferences and new devices to be added to the system in the future.

6.4.1 Introduction

So far we have specified how to use AUIT tags to create an AUIT page and display interfaces on the screen of a device. To be able to let an AUIT page work properly for a device, there are several variables that an AUIT page has to know.

- User type if more than one type of user can access the system.
- Tasks that people perform if various tasks can be performed.
- Device types used to browse an AUIT page.

We have demonstrated how to handle the first two variables previously. Now we need to discuss how to let the system know the users preferred screen size and some other information about that device, such as the text size, etc.

The most efficient approach to solve this problem is to have related data stored in database. That information needs to be stored before the application is used. Therefore a configurable interface needs to be designed before developer starts writing the application.

As mentioned previously, the `<AUIT:template>` tag has functionalities to detect user and device information when a user uses a device to browse an AUIT page. Once we know those values, the built-in function provided by `<AUIT:user>` and `<AUIT:device>` custom tags will decide whether to display the content between the start and end tags. We will illustrate how this information is stored into database in the following section:

6.4.2 Preference-based interface configuration

Adaptation may also be required due to particular user preferences. The reason may be to add new category of devices, users or tasks; information that is unknown at development time.

We achieve this purpose by providing a programmatic interface to access and modify them. Additionally, a user preferences component can be used which allows the user to specify the user interface related to preference information.

Some preferences may be system default, such as device MIME or maximum window size. Others can be obtained from the user interface aspect information input by user. Which preferences are variable & which are default is set by the developer or user. There are usually some set requirements when web developers build an application. We could use the following mechanism to capture end user and device information.

6.4.3 Examples

In our car site, if we allow for three accessing devices, then we need to enter the relevant information about those three devices. The following E-R diagram is an example for storing device and user preference data in a database.

We will describe the process if a user uses one of the devices intended to browse our application.

- If user logs in successfully, we are able to identify the user. Then we check if this user has preference in the “UserPrefs” table. If there is a “preference” record related to this user, then use the values, such as preferred screen size, to display the interface.
- If the identified user has no “preference” record in “UserPrefs” table, the browser will use the default information stored for the specified device to output the result.

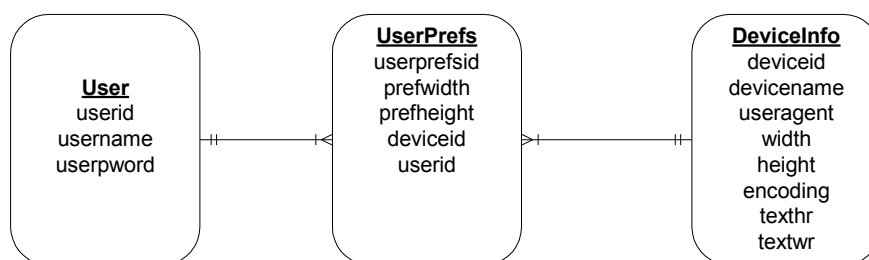


Figure 6.27 E-R Diagram of system required data

For example, using IE 5.0, PDA and Nokia browsers to open the interface page: After entering the data and clicking submit button, the following rows can be inserted into database.

Table 1 : DeviceInfo

deviceid	devicename	useragent	width	height	encoding	textr
1	IE5.0	Mozilla/4.0	1024	800	text/html	1.0
2	Nokia	Nokia-WAP-Toolkit/2.1	80	60	text/vnd.wap.wml	0.8
3	PalmOS	PalmVII	160	160	text/vnd.wap.wml	1.5

Table 2 : User

userid	username	userpassword	...(other info)
1	wendy	1234	...

Table 3 : UserPrefs

userprefsid	prefwidth	prefheight	userid	deviceid
1	800	600	1	1

By using the following interfaces, the user can input the device information, and the back end program will capture the relevant user agent data using web application functions. This data is inserted into the “useragent” field of the “DeviceInfo” table. Other fields have to be specified by the user.

Device Information

Device Name

Device Width

Device Height

Encoding

Text Ratio

User Preference

User Name

User Password

Preferred Height

Preferred Width

Figure 6.28 Sample interfaces for input system data

The above interfaces are using IE5.0 and “Nokia WAP Toolkit” browsers. When the user hits the submit button, the corresponding browser user agent information along with the input information are inserted into the database. So whenever the specific user accesses the site, the related device information can be fetched from the database by comparing the user agent data “fetched” with the user agent in the database.

6.4.4 How to use these interfaces

Before we start to use these interfaces, developers need to create several tables in database as we have shown in last section.

These tables are very general. In most of cases it is adequate for capturing the information that the system needs. But developers could have various tables depending on the system they are developing. After those tables have been developed, the developer can use the relevant devices they want their application to work on to browse these configuration interface pages and enter device-specific information they want their application to use.

6.5 Summary

In this chapter, we have explained how to implement the adaptable approach (AUIT). To achieve this we have created a set of JSP custom tags called AUIT tags. We have also reviewed some of the examples provided for developing car site in chapter three, this time using AUIT tags.

Chapter 7 - Case Study

7.1 Introduction

In this chapter, we will use the adaptable approach and the AUIT tags that we created in chapter six to develop a job management system. This system requires a variety of thin-client UIs, mostly for task, user and device adaptation.

7.2 Requirement Specification

A big company typically has an Information System that provides functions to manage the jobs that are assigned between staff. An organization intends to build a job management system to co-ordinate staff work. This needs to provide a variety of functions allowing staff to create, assign, track and manage jobs within an organization. Users of the system include employees, managers and executive management. Key employee tasks include login, job creation, status checking and assignment. In addition, managers and executives maintain departmental information, employee records and associated data. All of these user interfaces need to be accessed over an intranet using multi-device, thin-client interfaces i.e. web-based and mobile user interfaces.

7.2.1 User Requirements

This approach makes the system platform location-independent and enables staff to effectively co-ordinate their work no matter where they are. For example, a manager might like to view their required tasks and check on their employees work schedules using either a PC or a mobile device while they are outside the company. Authorised managers can also add new staff or users to the system whilst outside the office. In the following, we provide the detailed requirements:

Office Manager

1. Add a new department to the system or edit existing departmental data
2. Add a new management position or edit existing data

Departmental Manager

- 1 Add/update/edit Employees
- 2 View all jobs within his department
- 3 Authorised managers can view all employee information and organization data.

All Employees

1. Employees can set up their own login name and password, but their information needs to be preloaded, input by the manager in the first instance.
2. All employees can view all jobs assigned to him/her.
3. Employees can assign jobs to down-line employees or post a task to all or any other employees.
4. Anyone can reassign jobs to others if that job not within his job scope
5. Users can view all jobs related to them by accessing the system via a mobile phone

7.2.2 Use Case Diagram

These interactions that we have described are outlined in the use case diagram

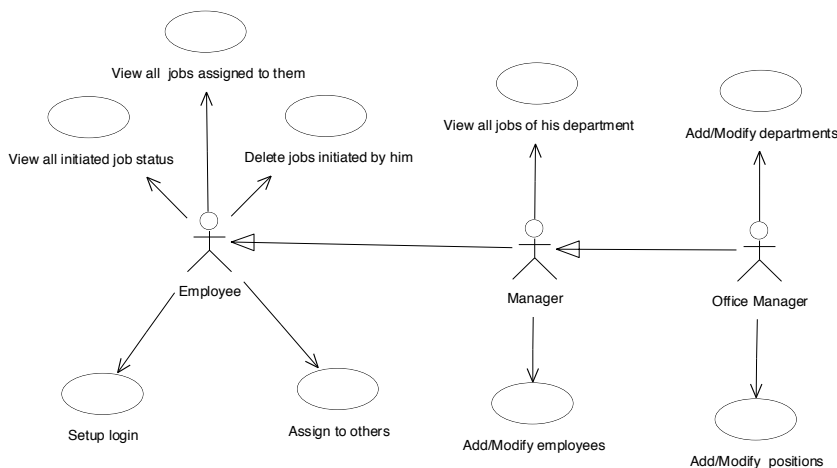


Figure 7.1 Example use cases of the job Management System.

7.3 Case Analysis and Design

In this section, we present the analysis and design of our application.

7.3.1 Object-Oriented Analysis (OOA)

The first important technical design decision is to identify the application objects. We will need data sources objects, business logic objects and number of JSPs to present the data.

We will use the tag library introduced in Chapter 4 to enable use of the adaptable approach where possible.

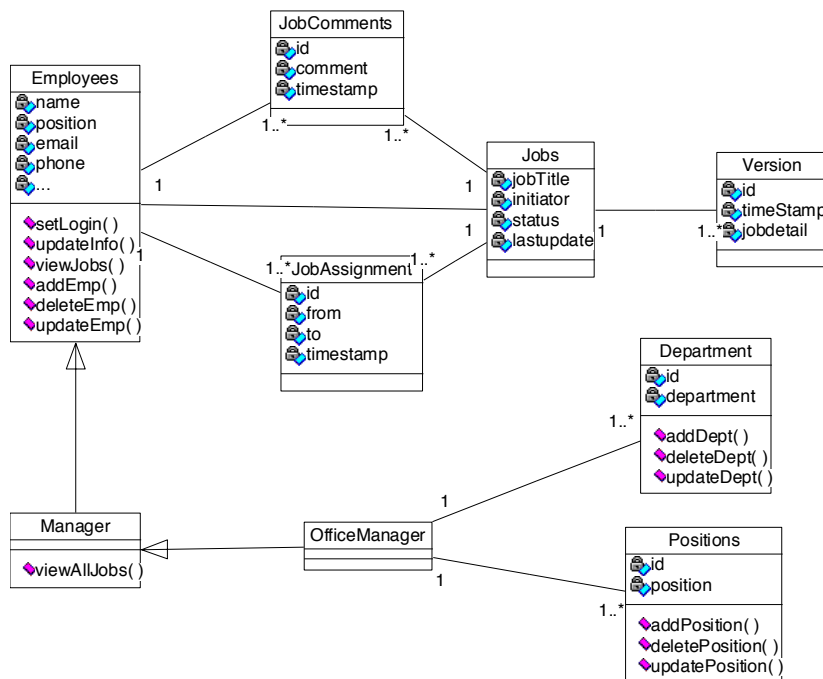


Figure 7.2 Class diagram of Job Management System

Figure 7.2 show class diagram presents an object-oriented analysis. The OOA diagram represents the conceptual view of the main system/application. From Figure 7.2, you can see the static structures of the system and inter-related objects that our system is comprised of. The relationships between objects are generalization and association. For example, “Manager” object generalizes to “Employees” object – the

manager can do all the tasks that general employee can do. “Employee” object and “Job” object are associated with “JobAssignment” object.

7.3.2 Software Architecture Design

In order to develop a new approach to build adaptive, multi-device user interfaces for web-based information systems, we need to consider the most suitable architecture. Our Job Management System adopts the following four-tier software architecture illustrated in Figure 7.3. Clients can be desktop and laptop PCs running a standard web-browser, mobile PDAs running an HTML-based browser or WML-based browser, or mobile devices like pagers and WAP phones, providing very small screen WML-based displays. All of these devices connect to one or more web servers (the wireless ones via a wireless gateway) accessing a set of AUIT-implemented screens, or web pages. The AUIT pages detect the client device type, recalls the user associated with the web session and tracks the user’s current task; typically by analysing which page(s) the current page has been accessed from.

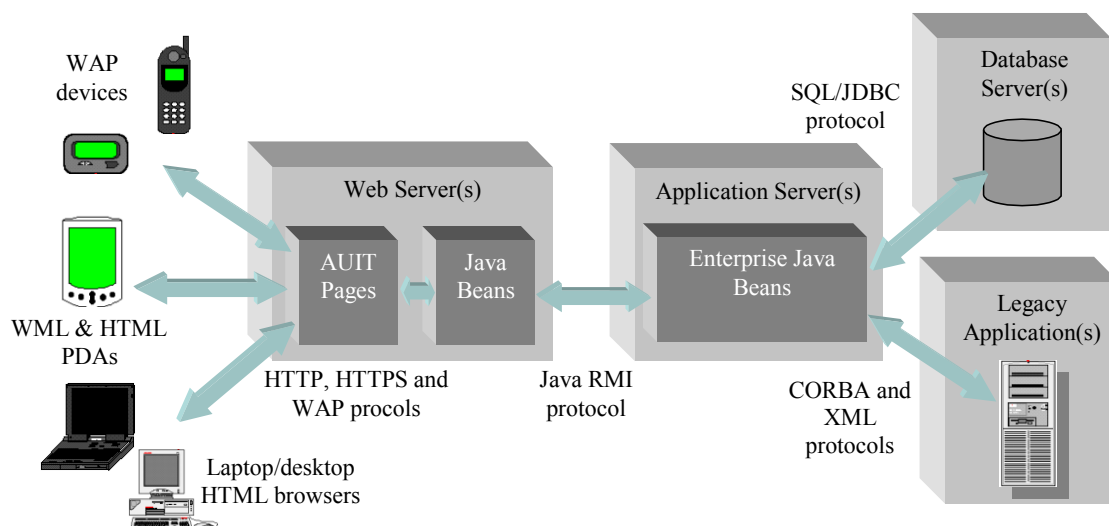


Figure 7.3. Our Four-tier web-based information system software architecture.

This information is used by the AUIT system to generate an appropriately adapted thin-client user interface, their current task context and their display device characteristics. AUIT pages contain Java code scriptlets that can access JavaBeans holding data and performing form-processing logic [Fields and Kolb, 2000]. This web

server-hosted JavaBeans communicate with Enterprise JavaBeans which encapsulate business logic and data processing [Vogal, 1998]. The Enterprise JavaBeans make use of databases and provide an interface to legacy systems via CORBA and XML.

7.3.3 Object-Oriented Design (OOD)

We now need to determine how the specification described in OOA will be translated into OOD designs after determining the software architecture for our system. Figure 7.4 shows the OOD diagram in which we can see that OOA objects are split into appropriate OOD classes that make up our programs and comprise the architecture we have designed.

We are going to use the Tag library as discussed in detail in Chapter four. The following discussion assumes familiarity with their use, so please refer back to chapter five and six if necessary.

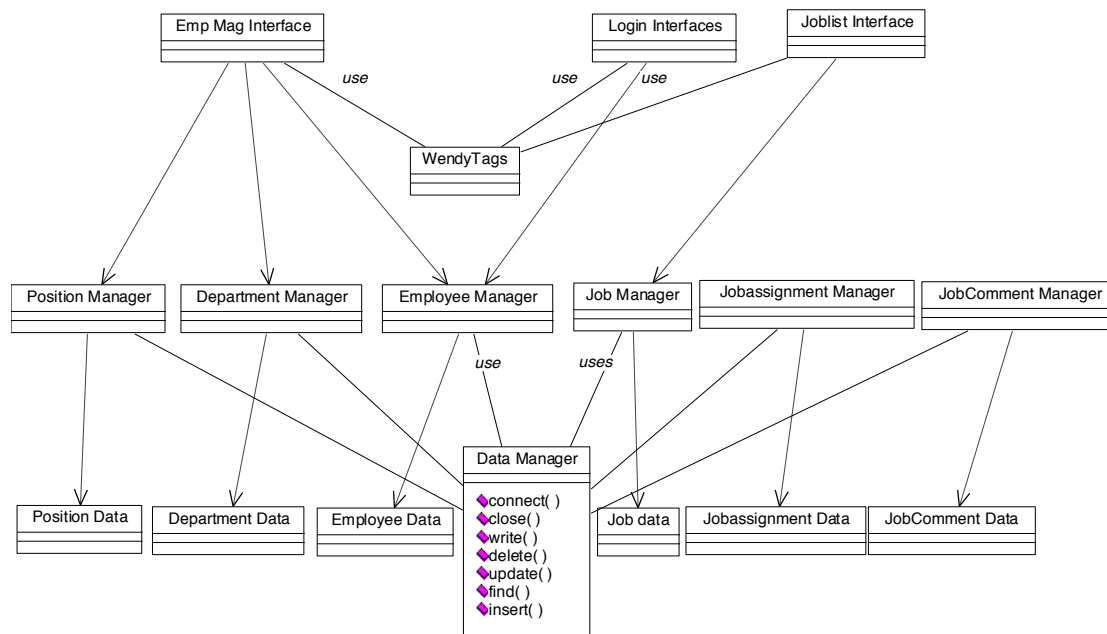


Figure 7.4 OOD Diagram

7.3.4 User Interface Design

In some of the key job management screens users are required to interact with the system to achieve tasks such as creating jobs, viewing job details, viewing summaries

of all assigned jobs and assigning jobs to others. These interactions are outlined in the use case diagram in Figure 7.1.

JoblistInterface and JobdetailInterface

The JoblistInterface shows a list of the jobs related to a particular user based on the login information. All the information listed on this page came from the “Jobs” table in the database. Three types of tasks in JoblistInterface that the user can perform all lead the user to JobdetailInterface. The JobdetailInterface will display different layouts based on the link that the user accessed the page from.

- 1) When the user clicks the link on the “Job Title” column, the next window will be shown displaying the detailed job description. The user can leave comments after they have done the job.
- 2) After they click “Assign New Job”, the second window will be shown. The user should be able to assign a new job to any other person.
- 3) If the user thinks that job is not for them, user can click the “Reassign To” link column to assign this job another person.

For example, in Figure 7.5, the job listing screens (1) for managers and employees are very similar, but management staff can see additional buttons and information fields. Sometimes the job details screen (2) has buttons for modifying a job (when the initiator is doing job maintenance) but at other times not (when the initiator is doing job searches or analysis, or the user is not the initiator). Sometimes interfaces are accessed via desktop PC web browsers (1 and 2) and at other times the same interface is accessed via a WAP mobile phone, pager or wireless PDA browser (3 and 4) when the employee wants access job information whilst away from their desktop or unable to use their laptop.

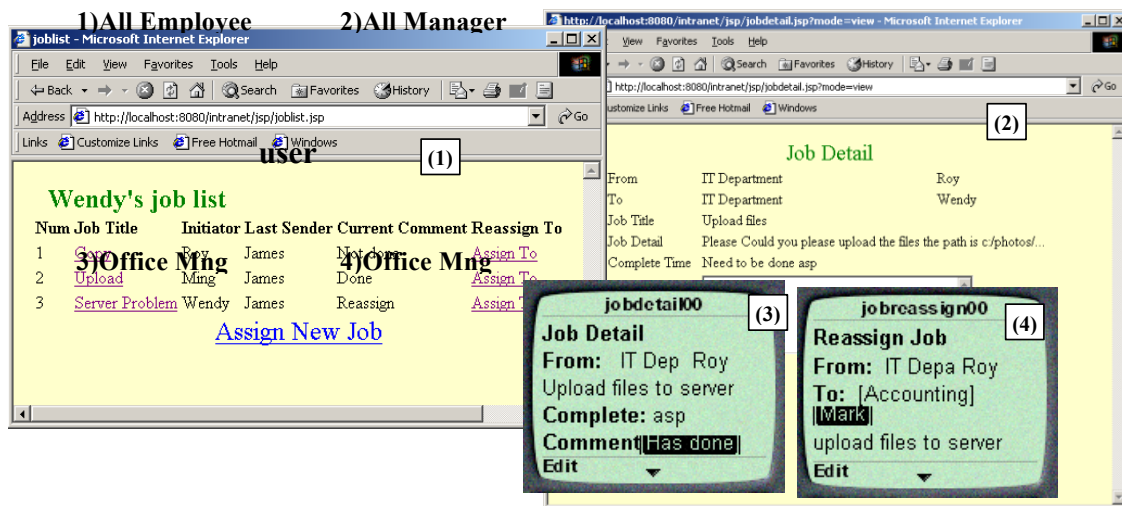


Figure 7.5 Examples of adaptive Job Management System screens.

The JobdetailInterface is a task-based interface. The page will have different layouts based on the task that user has performed and the page it came from.

AddnewInterface

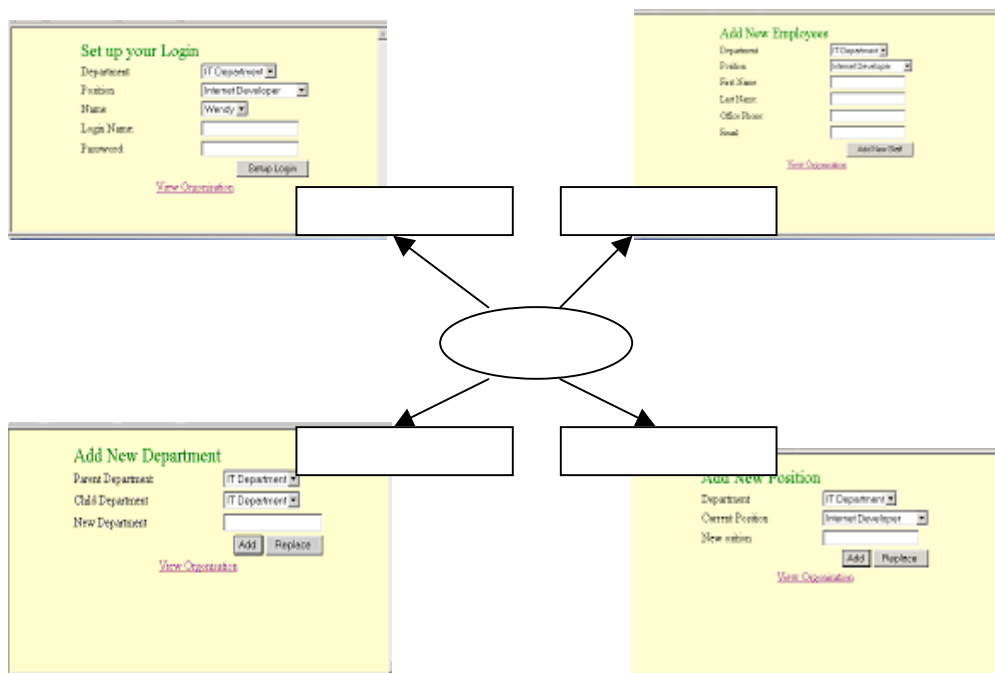


Figure 7.6 User base Interface for job manage system

Figure 7.6 shows a typical user-based interface. The four screens will serve different users, but with the same page.

- 1) All Employees can set up their own login information.
- 2) All Managers can add new employees within their departments.
- 3) The Office Manager can add New Departments to the system.
- 4) The Office Manager can add New Positions to the system

7.4 Implementation

Based on our design, we will implement our application by creating a set of AUIT pages using AUIT tags. Some of the thin-client, web-based user interfaces that our job management information system needs to provide are illustrated in Figure 7.5. Many of these interfaces need to “adapt” to different users, user tasks and input/output web browser devices.

7.4.1 Database Design

Before proceeding with our implementation we should prepare the database table storing the application-required data.

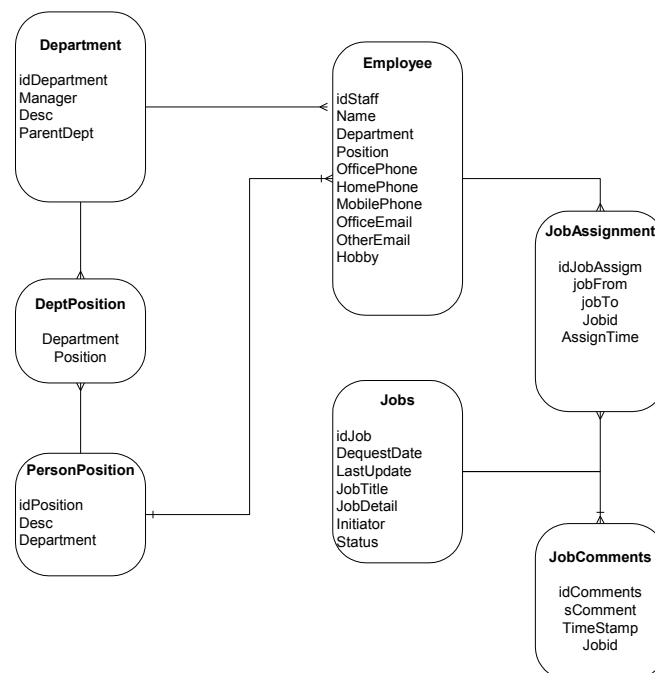


Figure 7.7 E-R Diagram (1)

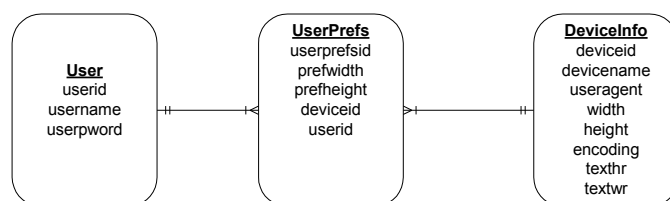


Figure 7.8 E-R Diagram (2)

The ER diagram in Figure 7.7 illustrates the application oriented data requirements. Figure 7.8 below indicates stored user-device data, user preferences, etc.

In order for the application to be used by a variety of devices, we need several other system-required tables as illustrated in chapter six. In chapter six, we introduced “Device”, “User” and UserPref” tables to solve the problem of different applications sometimes having different requirements. Because all our users are employees within the company our “Employee” table can replace the “User” table in Figure 7.8. We have specified the devices that employees can use, so we have the relevant device information inserted into the “Device” table by using specific interfaces such as those introduced in Chapter six.

7.4.2 Implementation Steps

The steps that developers should take to implement this application are as follows

- Set system environment and deploy AUIT package
- Prototype the applications using the AUIT tags and do business logic programming
- Test the result.

7.4.3 System Environment

The first step for applying AUIT tags is to check the specification on the tag library. We have provided a specification for the tag library and the AUIT custom tags have been specified with fairly straightforward explanation of the tag names and related attributes. Some of the attributes are mandatory so the user should be careful. The

users need to read the specification carefully to fully understand the functionalities that the tags provide.

As with the traditional approach we can start by installing the web and database servers. We can use Tomcat as our web application server again in order for our application to make use of the AUIT packages and approaches that we developed previously.

The last remaining file required to implement this application is the WAR's web.xml file. This has two important jobs: it must let the JSP engine know where to locate the tag library descriptor needed to describe the tag library and it must set the application's index page to "index.jsp".

Figure 7.9 shows the system development structure and part of code that we need to specified on "server.xml".

System Deployment	Part of "server.xml"
Tomcat -bin -conf -webapps -carsite -jobmanage -jsp other jsps -interfaces - WEB-INF -classes -tagext -beans -tlds AUIT.tld - web.xml -...	<pre> <Context path="/intranet" > docBase="webapps/intranet" crossContext="false" debug="0" reloadable="true" > </Context> </pre>

Figure 7.9 System Deployments

7.4.4 Implementing the Applications Using the AUIT Tags

By using the approach and AUIT elements that have been developed, the first step of implementation is to input the user and device information into the system. Once completed, the system “knows” the users and devices that the system would provide to. By using the interface that we have provided in previous chapter, we can enter the device information.

7.4.4.1 Implementing “JoblistInterface”

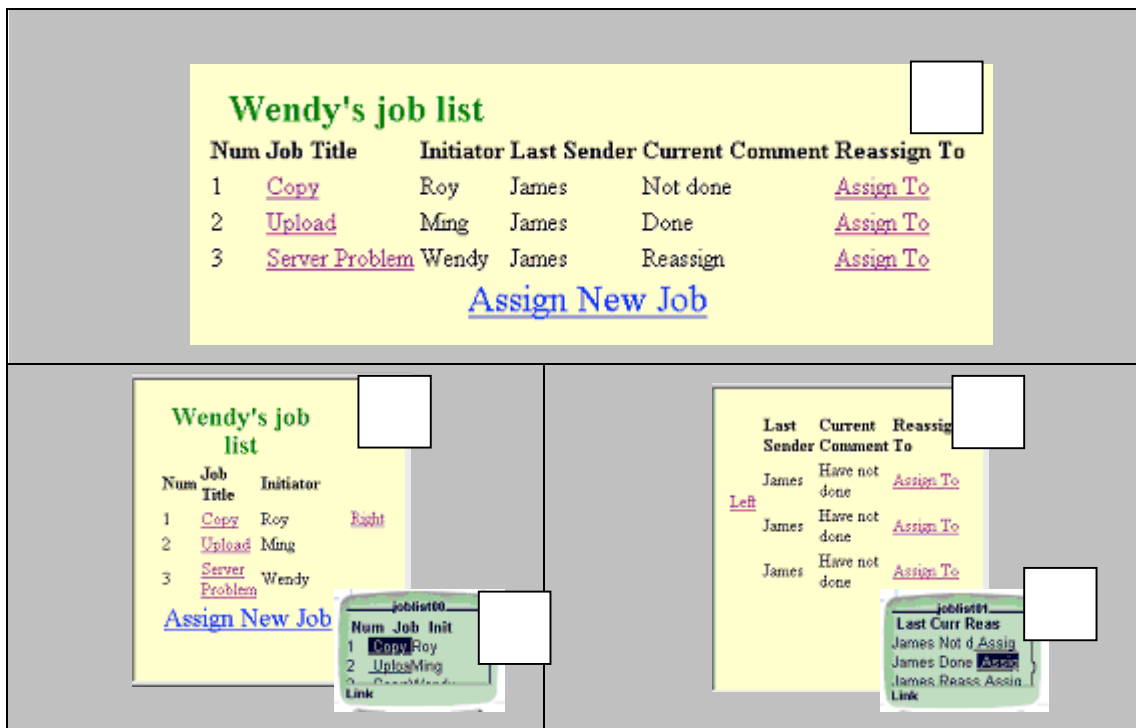


Figure 7.10 Examples of job listing screen running on multiple devices.

Figure 7.10 shows examples of the joblisting screen being displayed for the same user in a desktop web browser (1), mobile PDA device (2 and 3) and mobile WAP phone (4 and 5). The web browser can show all jobs (rows) and job details (columns). It can also use colour to highlight information and hypertext links. The PDA device cannot show all job detail columns, so additional job details are split across a set of horizontal screens. The user accesses these additional details by using the hypertext links added to the sides of the screen. The WAP phone similarly can't display all columns and rows. Links are added to access these. In addition, the WAP phone doesn't provide the degree of mark-up the PDA and web browser can, so buttons and links and colour are not used. The user instead accesses other pages via a text-based menu listing.

Figure 7.11(a) shows the logical structure of the “joblisting” screen using the AUIT tags introduced previously. The screen is comprised of a heading and list of jobs. The first table row displays column headings and subsequent rows are generated by iterating over a list of job objects returned by a Java Bean.

Figure 7.11(b) shows part of the AUIT Java Server Page that specifies this interface. The first lines in the JSP indicate the custom tag library (AUIT.tld) available and “JavaBean” components accessible by the page e.g. FunctionBean provides access to the database of jobs. The screen tag sets up the current user, user task and device information obtained from the device and server session context for which the page is being run. The heading tag shows the user whose job list is being displayed. The group tag indicates a group and in this example one with a specified maximum width (in pixels 8 per row) and alignment indication. The iterator tag loops, displaying each set of enclosed tags (each row) for every job assigned to the page user. The job list is obtained via the embedded Java code in the `<% ... %>` tags. The column values for each row include labels (number, jobtitle, initiator, etc) and links.

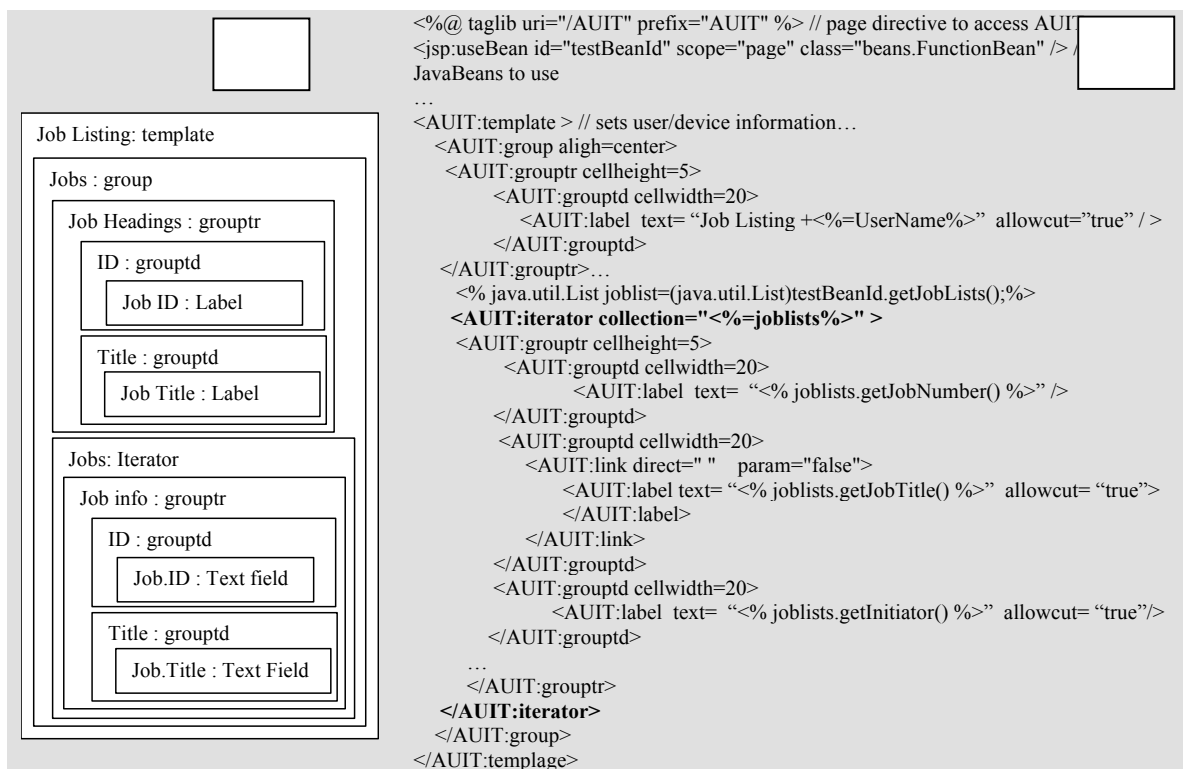


Figure 7.11 Logical Structure of the AUIT Description for the “joblisting”

7.4.4.2 Implementing “JobdetailInterface”

From the interface design section, we know that the user can perform three tasks on “JoblistInterface”. All the links on this page lead to “jobdetailInterface”, therefore we need to pass three different parameters onto the “jobdetailInterface” to differentiate the tasks that the user performed.

The interfaces shown in Figure 7.13 is mainly task based. The code are presented in Figure 7.13 explains the syntax that we used to create layout in terms of the value of “task”. This can control the task that user has performed and display proper content. Screens (a), (b) and (c) in Figure 7.13 show examples of the job details screen in use of desktop device, whereas (d), (e) and (f) are for mobile device.

1) Screen (a) and (d)

User can view detailed job, and make comments for that job when user has queries or needs to be assigned to another person. From these two interfaces, we can see we cannot use textarea element in devices other than a PC because of the screen size constraints, so we will convert “textarea” automatically into textfield type that allow the user to enter abbreviations.

Because this interface is going to display the job detail that other people assigned to the user, we need to display the “From” information in order and not display “To” information.

2) Screen (b) and (e)

People can perform “Assign a new job” task from the “joblist” page. After they click “Assign New Job”, they will see the view illustrated in Figure 7.13. With a bigger window size, people have plenty of room to describe the job details, but on a smaller window such as that on a mobile phone, the user should minimise the detail and use abbreviations where practical.

3) Screen (c) and (f)

People can reassign jobs to other people if necessary by using these two interfaces on different devices, Comparing the HTML & WML interfaces of screen (c) and (f) in Figure 7.13 demonstrates the differences between displaying the Job Detail screen on a PC screen and that of a Mobile phone. We have hidden the “Title” row on the mobile device in order to display more other the information in on window. Figure 7.12 shows the code needed to achieve this. The “Assign To” link from the job listing screen in Figure 7.13 lead user to job details screen in task context “job re-assignment”

```

<AUIT:device device="html">
  < AUIT:grouptr cellheight="5" >
    < AUIT:grouptr >
      < AUIT:layout bold="b">
        < AUIT:label text="Title"></ AUIT:label>
      </ AUIT:layout>
    </ AUIT:grouptr>
    < AUIT:grouptr colspan="2">
      < AUIT:layout bold="b">
        < AUIT:label text="Upload"></ AUIT:label></ AUIT:layout>
      </ AUIT:grouptr>
    </ AUIT:grouptr>
  </ AUIT:device>

```

Figure 7.12 Code Example

Part of the AUIT specification of the job details screen is shown in Figure 7.14. The screen encloses a form which, when the user fills out values, is posted to the web server for processing (done by the job_interface, JavaBean component). The heading is task-dependent – if the user is viewing job details, the heading is different to if they are assigning or adding a job. The ‘task’ attribute of the heading tags is used to determine which heading is shown. A table is used to achieve the layout. Some columns are common to all screens e.g. the left-hand side labels. Some rows are not shown for some screens e.g. the ‘From’ row is not shown if the user task is assign or new. Sometimes a different kind of form element is used e.g. if viewing a job, labels are used but when adding or assigning a job, some fields for the job have editable elements (text box, pop-up menu etc). If the user of the screen is not the job initiator, then the delete button is not shown.

view

assign

reassign

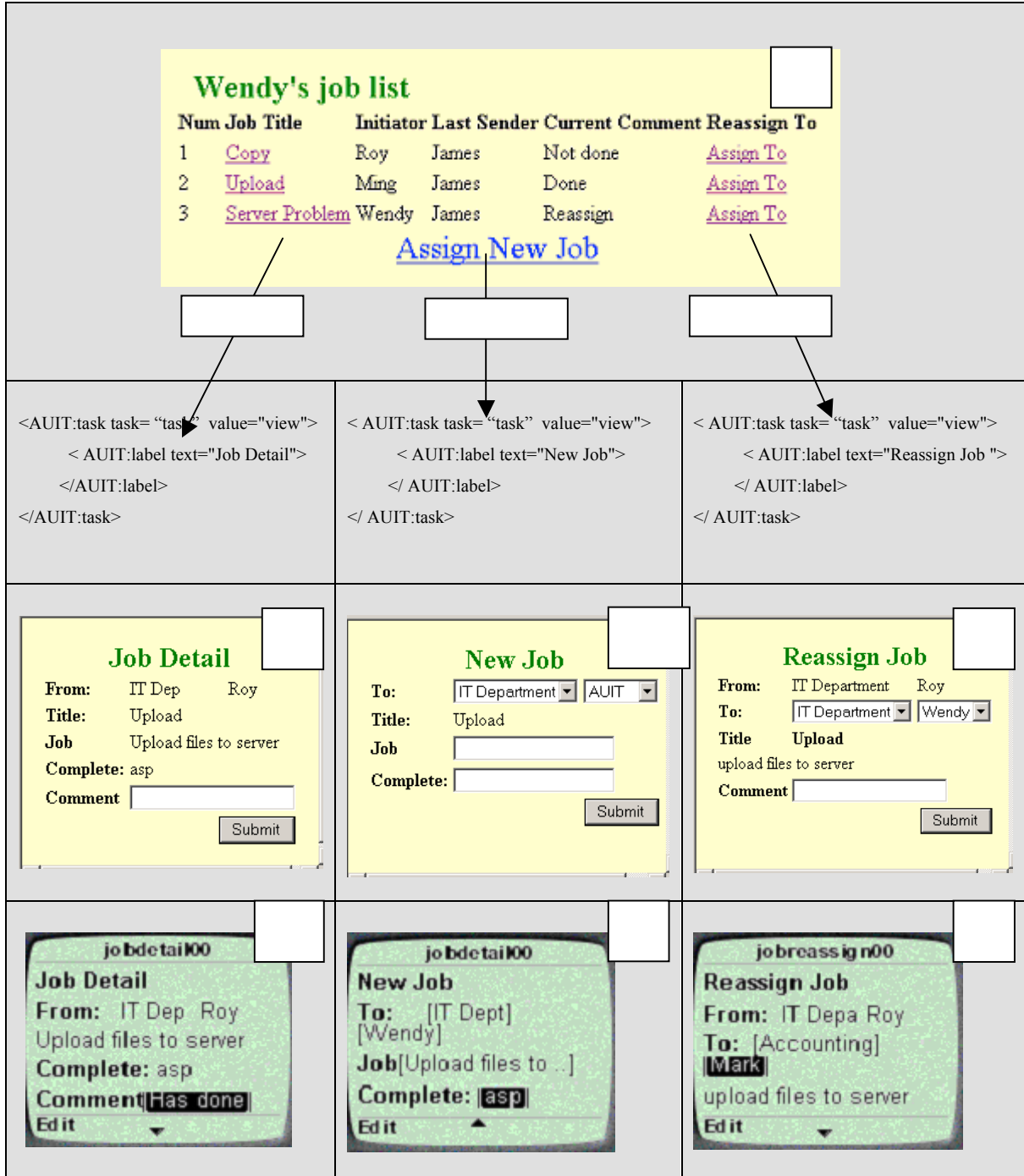
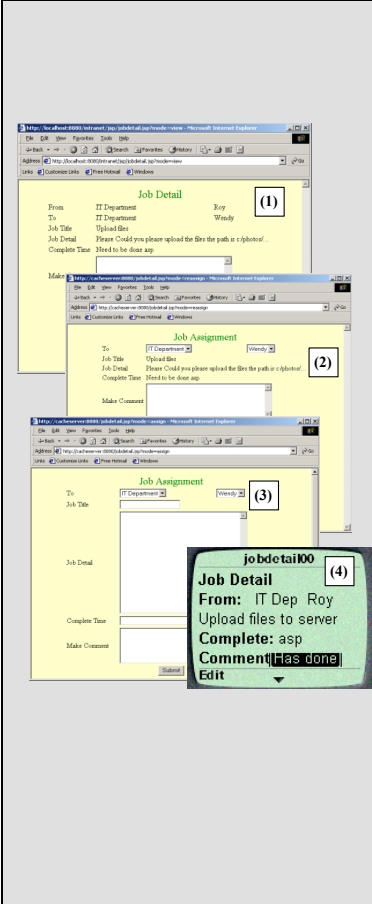


Figure 7.13 Implement Job list interface



The figure illustrates the AUIT description for a web application interface. On the left, three browser screenshots show different views of the 'Job Detail' and 'Job Assignment' pages, with numbered callouts (1, 2, 3, 4) highlighting specific UI elements. On the right, the AUIT description code is shown, demonstrating how tasks and devices are used to control the display of content based on the current view.

```

<%@ taglib uri="/AUIT" prefix="AUIT" %>
<jsp:useBean id='job_interface' class='jobs.JobDetailsInterface' />
<jsp:useBean id='job' class='jobs.JobData' />
...
<% job_interface.processRequest(request, job); %>
<AUIT:template name='job details'>
  <AUIT:group>
    <AUIT:form name='job details' action='job_details'>
      <AUIT:grouptr cellheight="5">
        <AUIT:grouptrd cellwidth="30">
          ...
          <AUIT:layout bold="b">
            <AUIT:task task="task" value="view">
              <AUIT:label text="From:"></AUIT:label>
            </AUIT:task>
            <AUIT:task task="task" value="assign">
              <AUIT:label text="To:"></AUIT:label>
            </AUIT:task></AUIT:task>
          </AUIT:layout>
        </AUIT:grouptrd>
        <AUIT:grouptrd cellwidth="30">
          <AUIT:task task="task" value="view">
            <AUIT:label text="IT Dep"></AUIT:label>
          </AUIT:task>
          <AUIT:task task="task" value="assign">
            <AUIT:device device="html" >
              <AUIT:select name="from" > ... </AUIT:select>
            </AUIT:device>
            <AUIT:device device="wml">
              <AUIT:textfield name="from" type="text" ></AUIT:textfield>
            </AUIT:device>
          </AUIT:task>
          ...
        </AUIT:grouptrd>
      </AUIT:grouptr>
    </AUIT:form>
  </AUIT:template>

```

Figure 7.14 Examples of Adapted Job Details and its AUIT Description.

From Figure 7.14, we can see how to use “task/device” tags to control content display. For instance, we want a UI component seen if the task value is either “assign” or “reassign”, then we can refer Figure 7.14 to code it.

7.4.4.3 Implementing Other Pages

Users can register on the Login page with a unique User ID and Password. After the system has verified the login details, the pages required by that specific user will be displayed.

From the above table, we can see that a page can have a lot of different views, depending on the tasks the page is required to perform. Creating this page using the traditional approach requires a vast number of conditional statements which are likely to complicate the logic of the entire code in the page. Our approach using tags to control the elements in the page will display relevant results efficiently without the need for potentially hard to maintain code.

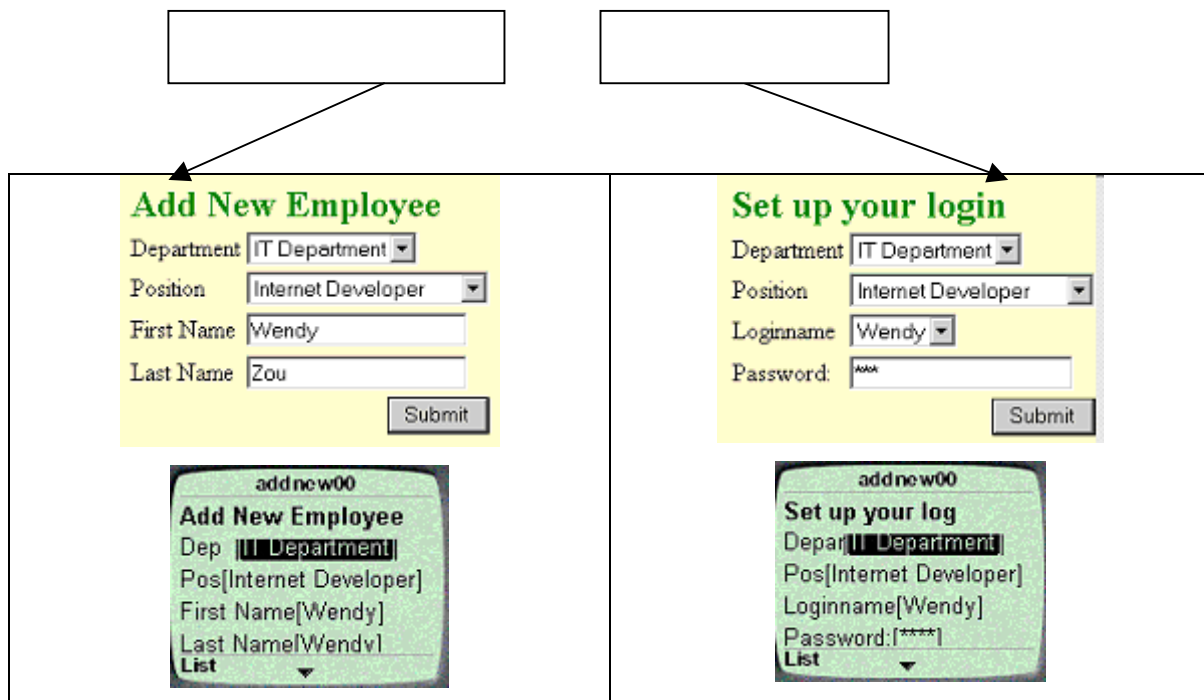


Figure 7.15 Task based Interfaces

7.5 Summary

In this chapter, we described an example application of the software architecture of our system and its implementation using the AUIT tags. Use of the Adaptable approach & AUIT tags to develop applications for multiple devices makes for cleaner code and reduced development time compared with traditional methodology.

Chapter 8 Evaluation

8.1 Introduction

In the previous chapters, we have developed a new approach to building adaptive, multi-device thin-client user interfaces for web-based information systems. User interfaces are specified using a device-independent mark-up language describing screen elements and layout, along with any required dynamic content (Java code). Screen element descriptions may include annotations indicating which user(s) and user task(s) the elements are relevant to. We have also provided a case study in chapter seven to demonstrate the development of a real world example.

In this chapter, we will evaluate the adaptable approach and the AUIT elements we implemented. In this evaluation we will discuss the strengths and weaknesses of our approach and discuss some areas that need to be improved.

8.2 Evaluation Topic

We have developed several systems with our technology, which have been evaluated by end users and are commercially deployable. We have built two substantial web-based information systems with AUIT technology: the job management system and an on-line car sales site. Each of these are commercially-deployable systems that have over two dozen AUIT screens, JavaBean, EJB application server components and database tables.

We have also built “hard coded” versions of these systems using conventional JSP technology - a commercial version of the on-line car retailing system and a commercial, in-house company job management system. Each of these systems has JSPs specifically built for different users, user tasks and display devices.

In this section, we will outline the evaluation criteria to be used by our testers. The main aim of our evaluation is to determine whether an adaptive approach is better than a non-adaptive approach and the strengths and weaknesses of the adaptable approach.

The evaluation will be divided into two main aspects:

- Examination of the adaptable approach and implemented elements from developers' point of view.
- Test the user interface developed using the adaptable approach from the end user perspective.

8.2.1 Evaluation from developers' perspective

In this part of evaluation, we will choose a group of people with web development knowledge or experience and ask them to evaluate our adaptable system based on the following criteria.

8.2.1.1 The Usefulness of Adaptable Approach

Web developers generally have predetermined opinions regarding the need to use an adaptable approach to develop a web-based application for various devices, based on their previous web development experience.

In chapter five, we have presented the motivations for using an adaptable approach and the practical experience of developing applications using "traditional approach". We want to know if, in the real world, developers "buy" the proposed benefits and would be willing to use an adaptable approach or AUIT tools to develop web-based systems.

1) Strengths of the Traditional approach

The main advantage of the traditional approach is a high degree of flexibility insofar as writing new code for each new system requirement means that the system's specific strengths can be fully utilised.

For example, a PDA has a touch screen for capturing input that is different from mobile phones and general desktop PC's, which use buttons or a keyboard. This feature allows a user interface to have more links on the screen because the user can use the pen to touch the screen and link to the other pages easily. Mobile devices in comparison direct users to different page by using softkeys, so that a simple one way link will be easy for this device. Writing for a specific device allows the developer to code functions that are unique to the PDA, making full use of the features of the device.

2) When to use an Adaptable Approach

When determining an approach, many trade-offs have to be made in order to produce a good system. These compromises will depend on the system & user priorities, such as development time constraints or specific content requirements. The Traditional approach favours complex, high quality graphics whereas the Adaptable approach provides advantages in programming efficient functionalities.

In some cases using the adaptable approach, developers have to compromise between some versions to improve productivity. A "One code fits all" approach will generally produce reasonable results but compromises will occur due to incompatibilities between the platforms, mark-up languages, screen size variations etc.

The layout using an adaptable approach generally cannot utilize all of the functionalities available on every platform as each interface layout is designed separately and specifically for it's respective device. By using the traditional approach the author can tailor the user interface with different priorities and goals based on the specific features of the device.

3) Examples

The job management system is a typical intranet application. The system is designed to improve staff efficiency and would not be seen by outside world. It requires a

functional rather than “pretty” user interface. For that reason the adaptable approach will be the better choice.

Therefore we want to evaluate the usefulness of Adaptable Approach from the developers’ perspective.

8.2.1.2 About Implementation of the Adaptable Approach

In chapter five, we illustrated the architecture of our adaptable approach and designed a high level language to achieve our adaptation requirements. We implemented our design using JSP custom tags in Chapter Six.

There are alternatives to implement our approach. Developers can implement this approach using technologies such as XML/XSLT, Microsoft ASP and so on.

We want to determine whether the JSP custom tag technology used to implement the approach was sufficiently substantial and comprehensive (relative to other technologies) to justify usage in web-based application development in the real world.

8.2.1.3 Ease of use of AUIT tags

As general requirement of any system, ease of use is most important. In our thesis, we have described an adaptable approach used to create User Interfaces and implemented our approach by creating a set of AUIT tags using JSP custom tags.

We are going to evaluate how easy it is to use the AUIT tags from the following perspectives, considering different adaptation purposes.

1) Adaptability of generic UI tags

In AUITs, we have created a set of generic elements (JSP custom tags) to generate the markup languages for rendering layout for specific devices. For example, if a device supports the WML, we will dynamically generate the WML for this device. We are

going to evaluate whether the adaptable approach can generate the Markup languages for specific devices satisfactorily.

2) UI Layout Adaptation

For the purpose of generating layout for specified screen size, we have created some algorithms embedded in the elements to make the system more powerful.

For example, as we described previously, we use “group, grouptr, grouptd” to specify the minimum dividable cell and use those elements with an algorithm to dynamically separate the content to fit properly on a small screen with links to relate them in separate windows.

We will evaluate this part by experiment and ask whether the testers are satisfied with the layout created by the UI layout control elements and embedded algorithm.

3) User and Task-based Interface Adaptation

In the traditional approach, to generate user and task based content developers usually have to embed a lot of conditional logic scriptlets in a page. This makes the code messy.

We have created some elements like “user”, and “task” in our adaptable approach and we need to evaluate the functions of those AUIT elements.

4) Device Adaptation

The adaptable approach has also provided ability for the system to generate device-based content so that the user can use them control the content, allowing suitable data to be displayed on specific devices.

For example, we can display long text on a desktop PC with an attractive graphics based layout, but specify short words to fit on small screen devices. We will ask our testers to evaluate and comment on this feature.

8.2.1.4 Configuration Ability Evaluation

Because the number and variety of devices used for Internet access is increasing constantly, we have provided a configuration engine allowing easy adaptation of the system to new devices. This configuration engine provides an easy to use interface allowing programmers to add relevant specifications for new devices, such as screen size and text rendering details.

The system also contains a feature which allows users to specify their preferred screen size.

For example, if the user wants to use a desktop browser with specified screen size instead of the default size, he or she can. We have provided a configuration interface for user to specify this detail. There are various ways to design similar configuration interfaces based on this idea.

We are going to evaluate whether the configuration interfaces that we have developed can capture enough relevant information.

8.2.1.5 Programming Productivity

The ability to develop and then deploy applications as effectively and as quickly as possible is also important. So we are going to evaluate productivity using AUIT compared to a traditional approach.

8.2.1.6 Code Maintenance Issues

The adaptable approach aims to simplify and speed the development process for developers and page authors alike and also to provide easy maintenance. Once an application has been developed, we are going to evaluate whether it is easy to maintain.

8.2.2 Evaluation from End User Point of View

In this part of evaluation, we will evaluate the user interfaces developed using an adaptable approach from end user point of view. To achieve this we have selected a second test group. This group are internet users who do not have programming knowledge. We will show them the two sets of user interfaces developed using adaptable approach and non-adaptable approach. We will ask them to complete a qualitative survey questionnaire and draw conclusions based on analysis of their responses. The evaluation will be based on the following criteria:

8.2.2.1 Ease of Use of User Interfaces

This is a comparative analysis of the interfaces (one developed using AUIT, the other using the traditional approach) from a perspective of ease of use and relative functionalities.

8.2.2.2 Acceptable Navigation Approach

The interface constructed using the adaptable approach uses links to relate windows on a small screen, which would appear as a single screen on a larger monitor. Do users find this navigation convention acceptable?

8.2.2.3 Screen Response

Speed is always the primary concern for an application for many users. There is no exception for our adaptable approach, so we are going to evaluate loading speed in this section.

8.2.2.4 Graphic Layout

Having an attractive graphic layout is one of the targets that many commercial web sites aim for. The AUIT user interfaces can achieve this in part; but it cannot compare with interfaces developed using traditional approach. We will provide screen shots from the two user interfaces and ask users for their comments.

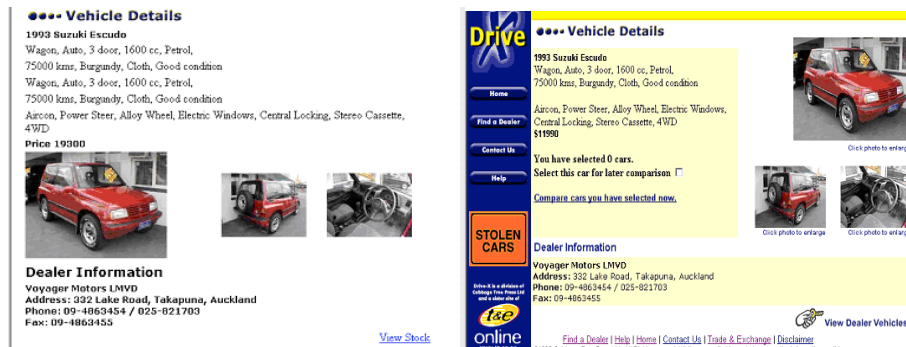


Figure 8.1 Page shots using adaptable approach and non-adaptable approach

8.3 Survey

In this section we will carry out a survey evaluating our system and summarize the general comments that the evaluators have made.

We intend to evaluate our approach and the AUIT tags from two perspectives. Firstly we will evaluate the adaptable approach and usability of AUIT tags from developer's perspective and we will measure the degree of satisfaction of the resulting product from the end user's point of view.

To achieve this we will use two test groups. One is comprised of web programmers, the other of web users. They will evaluate the technologies from their relative perspectives. All evaluators are required to read provided tutorials and complete the questionnaire that is specifically relevant to their field.

	Group One	Group Two
Number of Evaluators	6	10
Tasks to Fulfill	1) Read Introduction and Tutorial and become familiar with the AUIT tags 2) Install the system for testing 3) We have provided sample user interfaces comprising of two page dumps using JSP. Then we will ask the evaluator to develop a similar interface using the AUIT tags. 4) Fill the provided questionnaire	1) Read Introduction and Tutorial. 2) Look at the pages displayed on different devices with various window sizes developed using either AUIT approach or traditional approach. 3) Fill the provided questionnaire about the AUIT product interfaces.

Figure 8.2 Evaluator and Tasks

The survey that we ask evaluators to complete comprises of a number of closed and open-ended questions. The survey questions cover all of the requirements that the adaptable system is supposed to achieve. The detailed tutorials and surveys for evaluators in both groups are provided in Appendix B. In the following, we provide some example questions from the survey.

For example, in order to evaluate the AUIT tags we provided a table with a list of the AUIT tags and asked questions such as “Do you think it is easy to use the following JSP custom tags to implement adaptable approach?” The evaluators can tick the most appropriate answer from a five point likely scale. We have also provided some subjective questions such as “Comment on how you find the automatic layout generation of “group, grouptr, grouptd” tags.

8.4 Evaluation Conclusion

We have run two empirical evaluations of our AUIT-based systems, with end users comparing the AUIT and hard-coded system interfaces and with web-based information system developers comparing the use of AUIT technology to conventional JSP technology.

8.4.1 Evaluation Conclusion From the Developers

During the evaluation, we have asked the developers to create several AUIT pages using the AUIT tags. In terms of their experience with the AUIT tags, they are asked to fill in our questionnaire. We will draw conclusions by analysing their comments.

8.4.1.1 Comment about the AUIT tags

Most of the developers found AUIT tags to be straightforward to use, much more powerful and easier than the conventional JSP technology for building adaptable user interfaces.

- 1) Comments about Generic AUIT UI Elements

The design of AUIT-based systems is radically different to user interface design of conventional web-based application interfaces.

The AUIT UI elements include buttons, images, etc. Evaluators found almost all of them easy to use. For example, firstly they worried that controls like `<AUIT:textfield>` represented a completely new set of elements they would have to master. But they soon found that although the elements are new, they're not difficult to learn because they pretty much map onto their HTML equivalents.

Two things that they have noticed are that it is relatively simple to use the tags that we have designed compared to HTML tags. For example, when they use HTML, they have to handle the following three types of input type, but our design combines them, making it easy to master. One simple control can provide the functionality of three, depending on how it's used.

HTML tags	AUIT tags
<code><input type="text" ...></code> <code><input type="password" ...></code> <code><textarea ...></code>	<code><AUIT:textfield type="text"...></code> <code><AUIT:textfield type="password"...></code> <code><AUIT:textfield type="textarea"...></code>

Figure 8.3 Comparison of Usage between HTML and AUIT Tags

2) UI Control Elements

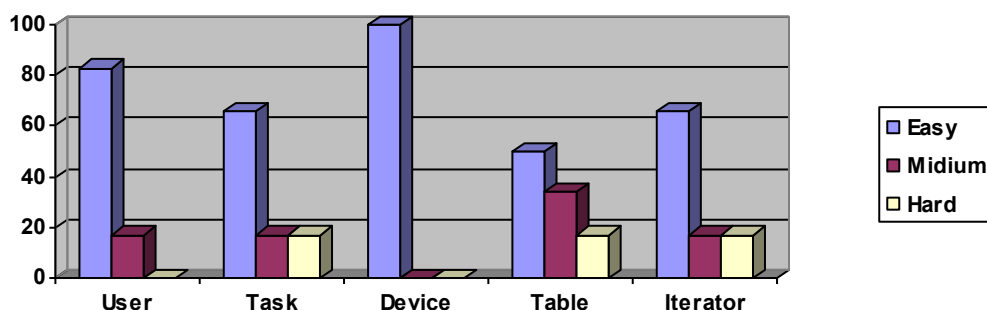


Figure 8.4 Bar Chart Diagram for Evaluation Result of UI Control Elements

From the bar chart shown in Figure 8.4, we found that most developers think the UI Control tags we have created are easy to use. Few of them have some comments about the usage of these tags. The comments are listed in the table shown in Figure 8.5.

Tags	Comment
User	Has limitation that cannot work on some of the devices that do not support cookies.
Task	The code looks a little complicated in following occasions where content is shown on several tasks. <pre><AUIT:task name= "task" value= "assign"> <AUIT:task name= "task" value= "view"> <!--Content--> </AUIT:task> </AUIT:task></pre> Others think this design is very good, "task" tag is easy to use.
Device	No comments returned.
Iterator	Some of the evaluators found it is not convenient that the "collection" attribute requires a value type of "java.util.List".
Table/tr/td	The functions provided by these elements are not powerful enough to handle complex situations such as "wrapped table".

Figure 8.5 Comments on the UI Control Elements from Evaluators

3) "Group/Grouptr/Grouptd" elements

These three elements are designed to control the "screen splitting" if necessary. They are part of the document structure and have to be specified. Most of people found these three elements are easy to use, just as easy as table elements of HTML. Some found they did have difficulty specifying the size of each cell, as we required. Figure 8.6 shows the evaluation result about this feature.

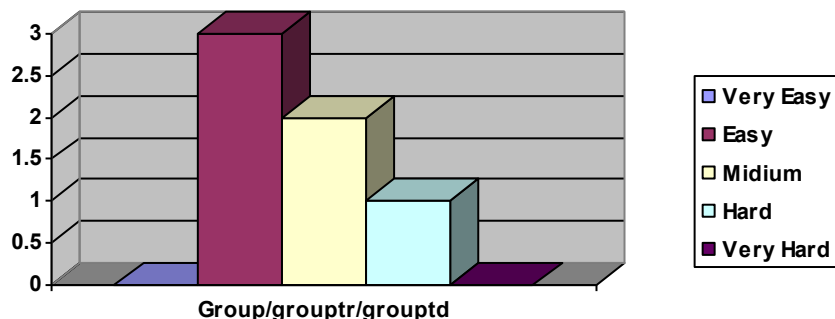


Figure 8.6 Bar Chart of Evaluation of "Group/Grouptr/Grouptd" Elements

AUIT groups provide limited ability to layout forms, which work well for WML and relatively simple HTML interfaces. Complex HTML layout for desktop browsers is difficult to achieve with the current AUIT grouping components. Estimating the amount of room rendered screen elements will take up, used by the screen folding algorithm to move some information to linked screens is difficult, as users may configure their device browsers with different default fonts and font sizes.

Evaluators found that it is sometimes hard to measure how much they needed to allocate for a cell that contains one or several elements. Because our approach required that the user has to specify the value of width and height of a cell, they needed to guess those values sometimes and adjust them later if they found it a problem when they tested the layout on-screen.

8.4.1.2 Functionalities of the AUIT Tags

Generally, they thought the AUIT tags that we have provided are sufficient for creating a simple adaptable interface. They found limitations with the AUIT tags; specifically, some of the functionalities that can be used for each technology cannot be fulfilled using the AUIT tags. For example, they found they cannot use frames for HTML.

8.4.1.3 Comment About Programming Productivities

The main advantage of the adaptable option is that it is relatively quick to develop an application for various human devices. The following table shows the steps that developers need to take for develop an application for various human devices using both approaches.

Non-adaptable approach	Adaptable approach
1.Learn WML markup language	1.Learn new AUIT custom tags
2.Write several versions for each page in order to serve several devices	2.Write one version for each page, and work for several devices

From the comparison on the above table, we assume that developers will spend similar time on the first step. Then for the second step, developers will spend more time using the non-adaptable approach than the adaptable approach. Overall time taken using adaptable approach will be significantly less. The greater the number of devices, the greater the time savings.

The other advantage is that developers write only one program, and work on the several devices with less worry about some complex aspects such as algorithms. The adaptable approach saves developers time.

8.4.1.4 Comment about Code Maintenances

The evaluators think the maintainability of code has been improved. There are two main reasons. Firstly the number of interfaces is reduced, requiring much less code than using the non-adaptable approach. The other reason is that the backend call to database and embedded code has been reduced. They observed that the code looks neater and would be relatively easy to maintain.

All evaluators report that they found our technology much easier to use and more powerful for building and maintaining simple adaptable web-based user interfaces than other current approaches. In fact, some found them better as they could change their device preferences and have the AUIT interfaces change to suit these, which is not possible with the hard-coded interface implementations.

The AUIT systems all have less than a third of the screen specifications than hard-coded systems. The screen specifications are much easier to extend and maintain as new data and functions are added to the systems, as only a single specification needs modifying rather than up to a half a dozen for some screens in some of these systems.

8.4.2 Evaluation Conclusion from End-user Perspective

Figure 8.7 shows the result evaluated from two criteria's from end-user perspective. On the following section, we will summarise the comments that they made on their questionnaire.

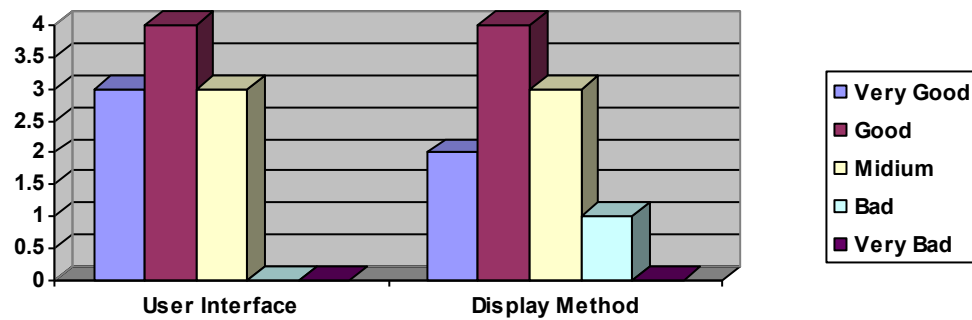


Figure 8.7 Evaluation Result from End-User Perspective

8.4.2.1 Comment about AUIT Interfaces

Most of end users don't care too much about the graphic layout as long as the site gives clear and adequate functionality. But some of them thought the graphic layouts on the AUIT interfaces are quite simple, not pretty enough and some of the presentation was considered a little bit "confusing".

Some content on desktop PC's cannot be translated to smaller screens, so needs to be chopped off. The interfaces that are developed using the adaptable approach sometimes be significantly different from original.

8.4.2.2 Comment about the Display Methods on the AUIT Interfaces

We have shown some extra navigation links on the interface to link to the next page if the screen is not big enough to display the entire information of an AUIT page. Most of the evaluators found no problem to accept and won't get lost. But some of them felt that there must be a better way to navigate, because the current navigation is cumbersome.

Overall, they think this approach is better than have a large amount of data displayed on the same screen, then scrolling the window to view them.

End users in general find the AUIT-implemented user interfaces almost as good from a usability perspective as the hard-coded ones.

8.5 Summary

In this chapter we have carried out an evaluation of the AUIT by developers and we have tested the adaptable interfaces that we have developed using the AUIT tags from the end users perspective.

We have also concluded our evaluation results and have generally discussed the strengths and weaknesses of our approach based on feedback.

Chapter 9 - Conclusion and Future Work

9.1 Introduction

In this chapter, we will conclude our thesis and present a summary of the main contributions of the thesis to the research field. An outline of the future work will be presented as well.

9.2 Contributions of the Thesis

In this thesis, we have developed a new approach for the development of adaptable, web-based information system user interfaces. Comparing with the previous work illustrated in “Chapter 2” and based on experience from using the traditional approach, we feel this thesis makes several contributions to the study of the adaptable user interfaces.

9.2.1 Easy to Use

Our approach allows developers to much more easily construct and maintain web-based user interfaces than other current approaches. The AUIT tags provided are similar to existing languages such as HTML, so that users don't have to learn a totally new language. This provides simpler syntax, because it can be used in an HTML-like syntax.

9.2.2 Creating High-level Languages

Another innovative feature of our approach is that it provides developers with a set of device-independent mark-up tags used to specify thin-client screen elements, element groupings, and user and user task annotations.

9.2.3 Powerful Adaptability

In “chapter 2”, we have examined previous work in this field. None of the previous studies provided the power as our system. Those systems generally could only achieve a small proportion of the adaptable functions that we have incorporated in our approach. For example, we have developed a novel automated approach for splitting large screens into parts to suit different display devices.

9.2.4 Easy Architecture

We have implemented our approach with Java Server Page custom tag libraries, making our system fully compatible with current J2EE-based information system architectures. AUIT technology allows developers to use all of the usual JSP and Servlet functionality in conjunction with the AUIT adaptable tags, meaning expressive power for building dynamic web applications is preserved without considering new architectures.

9.2.5 Extensibility

In this thesis, we have provided developers with a set of device-independent mark-up tags used to specify thin-client screen elements, element groupings, and user and user task annotations. Developers can easily extend our AUIT tags with more tags to make the system more powerful.

9.2.6 Good Productivity

This approach can improve the productivity of non-programmer content developers by allowing them to perform tasks that cannot be done with HTML. Moreover, by using those reusable elements, it will save development time for developers.

9.3 Future Work

There is a great deal of potential for future work on this topic. We have highlighted a number of areas that people are currently researching and the potential for future research. Our evaluations of AUIT have identified some areas for further research and development.

9.3.1 Developing a Tool to Create AUIT Pages

A lot of software packages, approaches and tools have been produced in order to save effort for developers. User Interfaces for applications, for example sun's forte, or JBuilder have made great progress, allowing Java developers to develop UI more easily. But developers usually have to write several lines to code a button on an interface before these tools start to be of use.

Although Web development tools are rapidly progressing, such as the latest versions of Interdev from Microsoft Visual Studio, or Dreamweaver from Macromedia Ltd., they still lag behind most graphical user interface (GUI) toolkits such as Swing or Visual Works Smalltalk.

For example, most traditional GUI toolkits provide layout managers in one form or another that allow layout algorithms to be encapsulated and reused. Window toolkits typically provide a layout mechanism that positions widgets in a container. For instance, AWT and Swing have layout managers, and Visual Works Smalltalk has wrappers. But we could not find any tools that will provide a mechanism that automatically lays out the web content so that it can fit in any size of screen. There are a lot of reasons, although the wide range of devices and lack of mature standard could be major stumbling blocks.

Dreamweaver is good tool for web application graphic designing. The user can visually design web site easily, without the need for knowledge of markup languages. But the result only works on one device-desktop PC. Designs made with Dreamweaver don't suit non-PC devices.

In our approach we need to hard code the AUIT pages. For example, we need to manually adjust the size of each cell in a "group".

A tool can be developed in the future to generate AUIT pages automatically. A researcher can develop new design methods for adaptable web application user interfaces, along with a GUI specification tool that will generate AUIT implementations from these graphical designs.

This tool could generate the basic AUIT page structure automatically and allow developers to drag some AUIT tags onto a panel and set up the properties of UI elements visually. During the drag and property setup, the relevant code for the AUIT page can be created. So that designing an AUIT page can be achieved visually by using such a tool and saved by the developer in the same places that we code manually.

Moreover, whenever an element is specified in the tool, some other configuration windows can ask if this element is displayed only for specific user, task or device.

The tool also can provide designers with the ability to work with logical structures rather than a fixed-format layout as in conventional user interface design.

9.3.2 AUIT Extensions

The implementation of our Adaptable Approach is just general idea, there is great potential for implementation with other technologies like ASP, etc.

9.3.2.1 Add More Functions for the AUIT Tags

Each tag that we have developed performs certain functionalities that we have demonstrated in “Chapter Five”. People can add more functions into each tag to make it more powerful.

For example, later work can improve the layout control in AUIT grouping constructs to give developers more control over complex screen layouts across display devices.

Current task adaptation support is quite limited and we are extending this to allow developers to use workflow-like information to support such adaptations.

Extending user preference control and device characteristics, like network bandwidth, will allow further detailed specification of interfaces adaptations.

9.3.2.2 Add More Tags

Extending our AUIT tags with more tags to make the system more powerful. Moreover, the AUIT tags that we have developed are clearly not sufficient for complex applications, so people could use this idea to develop their own tags for their own convenience.

9.3.2.3 Alternative Implementation Approach

There exists other possible technologies to implement our approach, (as we have specified in “Chapter Four”) aside of using JSP customer tags. People could make use our design idea and implement it with other possible technologies.

Although there’s a lot more to this approach than has been presented in this thesis, we aimed to give you a good starting point from which to explore further and begin to develop the adaptable approach concept further.

9.4 General Summary

In this thesis we have investigated the issues involved in building adaptive, multi-device thin-client user interfaces for web-based information systems. We began with a comprehensive review of the related research. We then overviewed relevant background technology. In order to address the strength of our adaptable approach over the traditional approach, we developed two versions of a car site, one using a traditional and the other with an adaptable approach; for later comparison.

In the review we examined some of the weaknesses and limitations of previous projects and drew some great ideas from them, which lead to our key requirements for developing the AUIT (Adaptable User Interface Technology).

We then proceeded with an analysis of requirements and developed a design to satisfy those needs. We implemented our design solution using the JSP custom tag technologies; a number of AUIT custom tags created to fulfill the different adaptation tasks issued in requirement. After implementation, we examined a case study which used the AUIT tags as the principal developing tool.

Finally we presented an evaluation of the adaptable approach and the AUIT tags from both developer and end-users perspectives and concluded the evaluation results. We have demonstrated the potential of AUIT for further enhancing adaptable user interface development.

Appendix A Bibliography

[Analyst Information] Analyst Information

["http://java.sun.com/products/consumer- embedded/allanalyst.html"](http://java.sun.com/products/consumer-embedded/allanalyst.html)

[Atlas Software Technologies]. Iftikhar Ahmed "Enabling Web Applications for Wireless Devices" by *Atlas Software Technologies, Inc.*

["http://www.sun.com/xml/developers/iftwireless/"](http://www.sun.com/xml/developers/iftwireless/)

[Annika Wærn] Annika Wærn "What is an Intelligent Interface?" (1997)

[Giles Davies] "Building WAP-enabled Applications with Jbuilder and Inprise Application Server." by *Giles Davies, Inprise/Borland UK*

["http://community.borland.com/article/images/26008/buildwap.pdf"](http://community.borland.com/article/images/26008/buildwap.pdf)

[Carroll, J.M.] Carroll, J.M. and Rosson, M.B. "The paradox of the active user." In J.M. Carroll (Ed.), *Interfacing thought: Cognitive aspects of human-computer interaction*. Cambridge: MIT Press/Bradford Books, 1987, pp. 80-111.

[Dewan, P. and Sharma, A] "An experiment in inter-operating, heterogeneous collaborative systems" In *Proceedings of the European Conference on Computer-Supported Co-operative Work*, Kluwer, pp. 371-390.

[Eric Cook] "An Introduction to Web Clipping and PQAs" *By Eric Cook*

<http://www.webreference.com/dev/webclip/index.html>

[Eisenstein, J. and Puerta, A. (2000)] Adaptation in automated user-interface design, In *Proceedings of the 2000 Conference on Intelligent User Interfaces*, New Orleans, 9-12 January 2000, ACM Press, pp. 74-81.

[eMobile Part 2] : "A Sample End-to-End Application Using the Java™ 2 Platform, Enterprise Edition - Integrating an Enterprise Application with Various Client Devices: Using Servlet, XML, and XSLT Technologies" by *Thierry Violleau MDE*

Enterprise Java Team Sun Microsystems, Inc. (Dec 2000)

“<http://java.sun.com/j2ee/white/eMobilePartII.pdf>”

[**Fields, D., Kolb, M. (2000)**]: Web Development with Java Server Pages, Manning.
Fields, D., Kolb, M. (2000);

[**Frank Rousseau**] Franck Roussear, J. Antonio Garcia-Macias, Jose Valdeni De Lima, and Andrzej Duda “User Adaptable Multimedia Presentations for the WWW” (visited at 12-09-2001)

[**Fox et al 1998, Palm, 2001**] Fox, A., Gribble, S. Chawathe, Y., and Brewer, E. (1998): Adapting to Network and Client Variation Using Infrastructural Proxies: lessons and perspectives, IEEE Personal Communications **5** (4), August 1998, 10-19.

[**Grundy et al**] J.C.Grundy, J.G.Hosking “Developing Adaptable User Interfaces for Component-based Systems”, Related Research

[**J2EE Edition**] Professional Java Server Programming J2EE Edition

[**Jean Vanderdonckt,**] Jean Vanderdonckt, Quentin Limbourg, “Synchronised Model-Based Design of Multiple User Interfaces. (Visited on 1/11/2001)

<http://www.cs.concordia.ca/~faculty/seffah/ihm2001/program.html>

[**jsptags(a)**] <http://coldjava.hypermart.net/servlets/wmltags.htm> last visited at Nov,22,2001.

[**jsptags(b)**] <http://coldjava.hypermart.net/servlets/escape.htm> last visited at Nov,22,2001.

[**Joseph L. Weber**] Joseph L. Weber, “Using Java 2 Platform” Special edition.

[**Korva 00**] Jari Korva ¹, Johan Plomp ¹, Petri Määttä ¹, Maija Metso ² “On-line service adaptation for mobile and fixed terminal devices” (2000)

[**Kules 00**] Kules. Bill, “User Modeling for Adaptive and Adaptable Software Systems”, pg , April (2000)

[**Queay H.Mahmoud**] Queay “WAP for Java Developers: Develop WAP Applications with Java Servlets and JavaServer Pages” (10/2001 visited)
<http://developer.java.sun.com/developer/technicalArticles/wireless/index.html>

[**Quseay H.Mahmoud Aug. 2001**] *August 2001* “Web Application Development with JSP and XML Part III: Developing JSP Custom Tags”
<http://developer.java.sun.com/servlet/> (visited on 12/05/2001)

[**Marsic, 2001a; Han et al 2000; Zarikas et al 2001**], Adaptive Collaboration for Wired and Wireless Platforms, IEEE Internet Computing July/August 2001, 26-35.

[**Nokia User**] “Nokia WAP Toolkit User’s Guide”,version 2.1, January 2001

[**Nokia Developer**] “Nokia WAP Toolkit Developer’s Guide”, version 2.1, (January 2001) “Using the Palm OS Emulator”

[**Palm OS**] “Programming Palm OS in a Nutshell”
<http://www.palmos.com/dev/tech/docs/palmos/Nutshell.html> (visited on 10/09/2001)

[**WAP Form Wireless**] “WAP Form Wireless Application Protocol WAP 2.0 Technical White Paper”. By www.wapforum.org *August 2001*

[**Qusay H. Mahmoud**] “WAP for Java developers - Develop WAP applications with Java servlets and JSP” by Qusay H. Mahmoud (visited on 10/10/2001)
<http://www.javaworld.com/javaworld/jw-06-2000/jw-0602-wap.html>

[**Rodden et al, 1998**] Rodden, T., Chervest, K., Davies, N. and Dix, A. (1998): Exploiting context in HIC Design for Mobile Systems, In Proceedings of the first Workshop on Human Computer Interaction with Mobile Devices.

[Steve Meloan] Steve Meloan, “The Jakarta Taglibs Project, *April 2001*”, (visited 11/2001)

<http://developer.java.sun.com/developer/technicalArticles/javaserverpages/JakartaTaglibs>

[Stephanidis, 2001; Grundy and Hosking 2001] Stephanidis, C. (2001): Concept of Unified User Interfaces, In *User Interfaces for All Concepts, Methods and Tools*, Laurence Erlbaum Associates, pp. 371-388.

[Tutorial SA and OOD] Tutorial #2: Software Architecture & OOD

http://www.cs.auckland.ac.nz/compsci335st/tutorials/Tutorial2/Tutorial2_2up.pdf

[Van der Donckt et al 2001; Petrovski and Grundy, 2001] Van der Donckt, J., Limbourg, Q., Florins, M., Oger, F., and Macq, B. (2001): Synchronised, model-based design of multiple user interfaces, In *Proceedings of the 2001 Workshop on Multiple User Interfaces over the Internet*.

Appendix B Evaluation Tutorial and Survey

1. Introduction

The purpose of this document is an evaluation of using AUIT tags (Adaptable User Interface Tags) to develop adaptable user interfaces for various human devices.

This document is comprised of three main parts. Firstly, we will introduce the basic intent and background of the evaluation. Secondly we provide a short tutorial about the adaptable approach and AUITs (Adaptable User Interface Tags) in the package that we developed for implementing the adaptable approach. Finally, we ask evaluators to fill out the questionnaires to issue their comments.

2. Background

This evaluation is part of the fulfillment of a Masters Thesis. In this thesis, we explore an adaptable approach for developing user interfaces for web based applications in a variety of human devices and compare it to traditional methods. To implement the approach we have used JSP custom tags technology to build a set of elements (AUITs) for web developers to use for developing web-based applications.

We intend to evaluate our approach and the AUITs from two perspectives. Firstly we will evaluate the adaptable approach and usability of AUITs from developer's perspective, and secondly, we will measure the degree of satisfaction of the resulting product from the end user's point of view.

To achieve this we will use two test groups. One is comprised of web programmers, the other of web users. They will evaluate the technologies from their relative perspectives. Evaluators are required to read tutorials and complete the questionnaire that is specifically relevant to their field.

3. Evaluation for Group One – Developers Perspective

This part of evaluation needs to be done by group members with software development knowledge or experience.

3.1 How to run the application

In this section, we provide a tutorial for the evaluation of how to use the AUITs elements to create user interfaces for various devices. The system environment that we used is as follows

1) JDK1.3/ Tomcat

Download the tomcat installation ZIP file and unzip this to “C:\tomcat”. Follow the instructions with it to start up

- Start a DOS command window
- Cd to C:\Tomcat
- Set JAVA_HOME to where you have your JDK1.3 installation
- Click bin\startup
- A new DOS window should open with Tomcat http server messages display.

2) Use Microsoft Access as database tool

The testing program “carsite” use an MS Access database called “carsite.gdb”. To run them, we first need to set up ODBC DSN name called “carsite” to point to this. To do this, go:

- Select Start|Settings|Control Panel on the Windows desktop
- Double-click ODBC Data Sources
- Click on Add
- Select MS Access Drive, then Finish
- Set Data Source Name to “carsite”
- Click Select and choose the “carsite.gdb” file.

3) Deploy the package

Suppose that we still use tomcat as our web server, under the tomcat folder, we can find the folder structure as in the table of column two,

The following is the deployment for the car site application.

The deployment for AUIT approach	The deployment for traditional approach
carsite -jsp -carsite -800x600html -200x100wml -100x100wml -interfaces -system -WEB-INF -classes -beans -tagext -tlds wendy.tld -web.xml	carsite -jsp -carsite -WEB-INF -classes -beans

Figure B.1 System Deployment

4) Testing Device

- IE 5.0 as browser to test the interface displayed on the General PC using HTML and with normal window size (800x600).
- Use mobile device emulator-Nokia Toolkit 3.0 to test the interface displayed on other devices using WML with small window size.

Download the version 3.0 of the Nokia Mobile Internet Toolkit at the Nokia web site <http://www.forum.nokia.com/wapforum/main/toolkit>. You can change the testing window size by setting “width” and “height” of the simulator display screen (in pixels), enter the desired number of pixels in the text entry box for “width” between 84 and 384) and “height” (between 96 and 512).

3.2 Tutorial

Firstly, evaluators need to spend some time familiarising themselves with the AUITs tags in the tag library, so they are able to use them to build an interface.

The following diagram shows the basic tags that we have developed in AUITs. The tag name is on the top of each box, and the bottom lists the attributes associated with that tag. The star sign in front of the attribute means that this attribute is required and must be specified when use this tag.

1) Page Flow Control Elements

The basic Each AUIT page should have the following structure.

```

<AUIT:tempate>
  < AUIT group >
    <AUIT:grouptr >
      <AUIT:grouptd >
        UI elements
      <AUIT:grouptd >
      <AUIT:grouptd >
        UI elements
      <AUIT:grouptd >
    </AUIT:grouptr>
    ...
  </AUIT:group>
</ AUIT:tempate>

```

	Indication
AUIT:Template	This element is used to construct the basic structure that basic HTML and WML file require, as illustrated in the above example

Figure B.2 “Template” Tag Description

The following elements are used for controlling the layout of the user interface. They are:

	Indication
AUIT:Group AUIT:Grouptr AUIT:Grouptd	These elements can be used the same way as “table” tag in HTML, and the attributes are displayed in the above diagram. These are compulsory elements, each JSP file need to have these tags specified, and should be used as in the above example.

The diagrams lists in the following show the name and properties of AUIT tags.

2) Form and User Input Elements

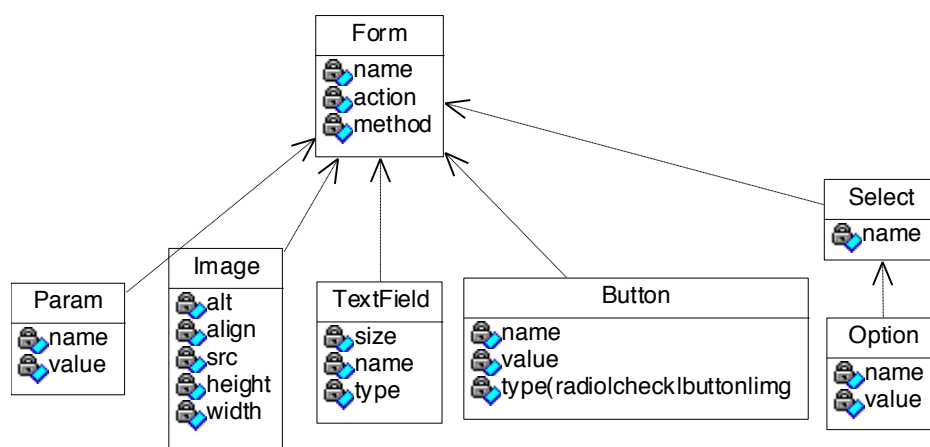


Figure B.3 Form and user input elements

3) Text and text format elements

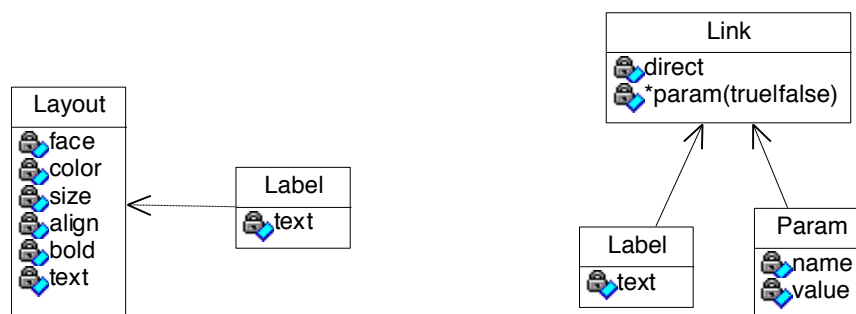


Figure B.4 Text and text format elements

The following table lists the main UI elements that can be used for building the user interface:

	Indication
AUIT:Select AUIT:Option	A select list specifies a list of options that the user can choose from. The usage of these two tags is exactly same as the tags in HTML.
AUIT:Break	This tag is simple as in HTML
AUIT:TextField	This tag can have three types, text, password and textarea, all of them can be used by JSP file for generating HMTL. The first two can be used for generating WML; the last one will be omitted when generate WML.
AUIT:Image	Image tag is used to display images on the html and wml file, it can display the proper format of image by detecting the device type and generate proper markup language that the device support.
AUIT:Button	It has four types (radio,check,button,img). It displays a button for interface.
AUIT:Link	Link can direct page to specified url attribute
AUIT:Label	It is used for specifying the text in a file, “allowcut” attribute is for specifying if the page author allow the part of text to be chopped off in order to displaying nicely on a small size window.
AUIT:Layout	This element is similar with the element “font” in HTML, but more powerful. User can specify the attribute as indicated in above diagram
AUIT:Form	The usage of this tag is used handle user input and event.
AUIT:Parameter	This tag can be used for pass parameters during pages redirect and refresh.

4) UI control elements

This type of elements are responsible for control the UI elements, decide if the UI elements will be displayed, and how they will be displayed. The following diagram shows the elements and the their attributes that I designed for different occasions.

Task and User can be used to control

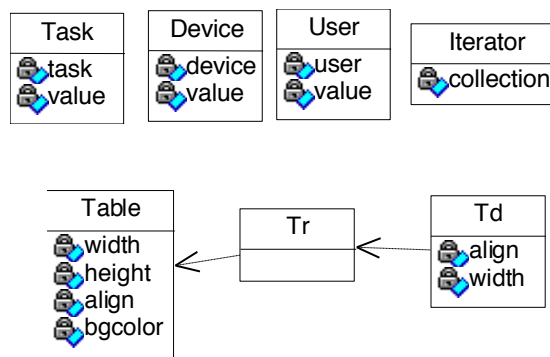


Figure B.5 UI control elements

	Indication
AUIT:User	It is used to control the user based content display. Only content that is user-specific can be displayed. Others will be filtered out.
AUIT:Task	It is used to control the task based content display. Only content that is task-specific can be displayed. Others will be filtered out.
AUIT:Device	It is used to control the device based content display. Only content that is device-specific can be displayed. Others will be filtered out.
AUIT:Table AUIT:Tr AUIT:Td	These elements are used to display the content in table structure, for HTML enabled devices.
AUIT:Iterator	This element is used to iterate the similar content in a page.

- Step two:

We have provided sample user interfaces comprising of two page dumps using JSP. Then we will ask the evaluator to develop a similar interface using the AUIT tags.

These two pages are part of a web site on which car dealers can advertise their stock and dealer information and web users can browse information from the site via the Internet. We have also provided some of the code for the pages and asked the evaluator to build the rest of the code to complete the page. The sample code relates to the usage of the tags. The evaluator can also refer to the sample code for other pages provided.

Page One-Search_car.jsp

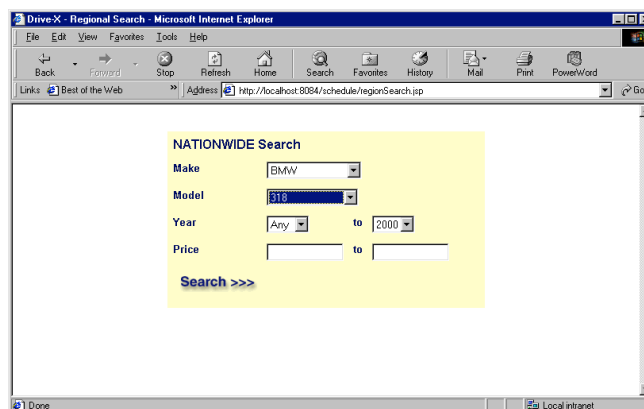


Figure B.6 Page shot for “Search_car”

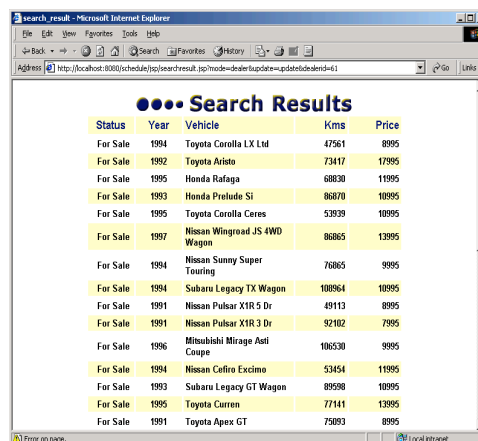
On home page of car site, we provide a list of region for user to select, and user can click any of the regions. When people select a region displayed on home page, and the

following page show up on which user can select certain criteria to search cars they like to browse.

This interface cannot be displayed fully on the mobile device. Therefore, we ask the evaluator to develop it using adaptable approach. Developers can use the elements we described on the above tutorial, and build the following interface.

Page Two-Search_result.jsp

The following page displays the car list based on the search criteria that user selected from the above page, then click search button. Again this interface is also a problem if we use device with small window size to display it. We ask the evaluators to develop this page using the adaptable approach.



Status	Year	Vehicle	Kms	Price
For Sale	1994	Toyota Corolla LX Ltd	47561	8995
For Sale	1992	Toyota Aritto	73417	11995
For Sale	1995	Honda Rafaga	68830	11995
For Sale	1993	Honda Prelude Si	86970	10995
For Sale	1995	Toyota Corolla Ceres	53939	10995
For Sale	1997	Nissan Wingroad JS 4WD Wagon	86865	13995
For Sale	1994	Nissan Sunny Super Touring	76865	9995
For Sale	1994	Subaru Legacy TX Wagon	100964	10995
For Sale	1991	Nissan Pulsar X1R 5 Dr	49113	8995
For Sale	1991	Nissan Pulsar X1R 3 Dr	92102	7995
For Sale	1996	Mitsubishi Mirage Asti Coupe	106530	9995
For Sale	1994	Nissan Cefiro Excimo	53454	11995
For Sale	1993	Subaru Legacy GT Wagon	89598	10995
For Sale	1995	Toyota Curren	77141	13995
For Sale	1991	Toyota Apex GT	75093	8995

Figure B.7 Page shot for “Search result”

- Step three

After the evaluators complete the pages, they can test them using IE and Nokia environment that we have set for them. They are then required to answer the following questions based on their experiment.

3.2 Survey Questions

Usefulness of the Adaptable Approach

From your web-based application developing experience, do you think it is advantageous to use an adaptable approach to develop user interfaces for various human devices?

1. Very useful 2. Useful 3. No preference 4. Not very useful 5. Not useful

Ease of adapt

- What technologies do you think would be easier to implement using this approach?

- Do you think it is easy to use the following JSP custom tags to implement adaptable approach?

	Easy to use	Not easy to use	Similar with traditional approach	Hard to use	Very difficult to use
Template					
Option/Select					
Break					
TextField					
Image					
Button					
Link					
Label					
Layout					
Iterator					
Table/Tr/Td					
Form					
Parameter					

- Provide additional comment

--

- Please indicate your opinion about the following tags

	Easy to use	Not easy to use	Similar with traditional approach	Hard to use	Very difficult to use
Group					
Grouptr					
Grouptd					

- Comment on how you find the automatic layout generation of “group, grouptr, grouptd” tags.

--

User, Task and device-based Interface Adaptation

- Do you think it is easy to use our user, device and task control approach to build user and task base user interface?

	Easy to use	Midium	Hard to use
User			
Device			
Task			
Table/td/tr			
Iterator			

- Comment on whether you think that the elements that we have designed are sufficient for developing web-based interfaces for various human devices?
-
-

Programming Productivity

- Do you think it has more productivity using the adaptable approach comparing using traditional approach?

1. Much more 2. More 3 Similar 4 Less 5 Much less

Maintenance

- Do you think it would be easy to maintain an application developed using adaptable approach than using traditional approach?

1. Much easier 2. Easier 3.Similar 4.Harder 5.Much harder

4. Evaluation for Group Two

In this part of evaluation, we will ask users their opinions of the user interfaces developed using the adaptable approach.

4.1 Tutorial

In this tutorial, we are going to provide user interfaces that are developed using adaptable approach. Those interfaces are also part of web sites on which car dealers can advertise their stock and dealer information and web users can browse information from the site via the Internet.

Car Display

The following shows three sets user interfaces developed using adaptable approach. This window displays a list of stock information of a car dealer.

The window size is 600*400 pixels on the following window, the complete information about a list of cars can be displayed on the window with this screen size.




Palm Emulator with 160x160 can display three cars per screen	Mobile with 90x100 can display one car per screen	Desktop with 800x600 can display 15 cars per screen																																																																																
		 <table border="1"> <thead> <tr> <th>Status</th> <th>Year</th> <th>Vehicle</th> <th>Kms</th> <th>Price</th> </tr> </thead> <tbody> <tr> <td></td> <td>1999</td> <td>Audi A1 Quattro</td> <td>90000</td> <td>\$49999</td> </tr> <tr> <td></td> <td>1999</td> <td>BMW 118</td> <td>9000</td> <td>\$33995</td> </tr> <tr> <td></td> <td>1999</td> <td>BMW 118i</td> <td>12200</td> <td>\$54990</td> </tr> <tr> <td></td> <td>1999</td> <td>BMW 118i</td> <td>8800</td> <td>\$51990</td> </tr> <tr> <td>Arriving</td> <td>1999</td> <td>BMW 120i</td> <td>55000</td> <td>\$23999</td> </tr> <tr> <td></td> <td>2000</td> <td>BMW 120i LASE</td> <td>2716</td> <td>\$90000</td> </tr> <tr> <td></td> <td>2000</td> <td>BMW 120i LASE</td> <td>5000</td> <td>\$90000</td> </tr> <tr> <td></td> <td>2000</td> <td>BMW 120i Car</td> <td>6500</td> <td>\$81990</td> </tr> <tr> <td></td> <td>1999</td> <td>BMW 120i SE</td> <td>41070</td> <td>\$69990</td> </tr> <tr> <td></td> <td>2000</td> <td>BMW 120i Touring</td> <td>9000</td> <td>\$79990</td> </tr> <tr> <td></td> <td>2000</td> <td>Chrysler Neon</td> <td>15004</td> <td>\$21995</td> </tr> <tr> <td></td> <td>1999</td> <td>Chrysler Valiant</td> <td>34000</td> <td>\$31995</td> </tr> <tr> <td></td> <td>2000</td> <td>Chrysler Voyager SE</td> <td>500</td> <td>\$49995</td> </tr> <tr> <td></td> <td>1999</td> <td>Citroen Xsara SX</td> <td>45095</td> <td>\$22995</td> </tr> <tr> <td>NEW</td> <td>2001</td> <td>Daciaun Lotus SE</td> <td></td> <td>\$19495</td> </tr> </tbody> </table>	Status	Year	Vehicle	Kms	Price		1999	Audi A1 Quattro	90000	\$49999		1999	BMW 118	9000	\$33995		1999	BMW 118i	12200	\$54990		1999	BMW 118i	8800	\$51990	Arriving	1999	BMW 120i	55000	\$23999		2000	BMW 120i LASE	2716	\$90000		2000	BMW 120i LASE	5000	\$90000		2000	BMW 120i Car	6500	\$81990		1999	BMW 120i SE	41070	\$69990		2000	BMW 120i Touring	9000	\$79990		2000	Chrysler Neon	15004	\$21995		1999	Chrysler Valiant	34000	\$31995		2000	Chrysler Voyager SE	500	\$49995		1999	Citroen Xsara SX	45095	\$22995	NEW	2001	Daciaun Lotus SE		\$19495
Status	Year	Vehicle	Kms	Price																																																																														
	1999	Audi A1 Quattro	90000	\$49999																																																																														
	1999	BMW 118	9000	\$33995																																																																														
	1999	BMW 118i	12200	\$54990																																																																														
	1999	BMW 118i	8800	\$51990																																																																														
Arriving	1999	BMW 120i	55000	\$23999																																																																														
	2000	BMW 120i LASE	2716	\$90000																																																																														
	2000	BMW 120i LASE	5000	\$90000																																																																														
	2000	BMW 120i Car	6500	\$81990																																																																														
	1999	BMW 120i SE	41070	\$69990																																																																														
	2000	BMW 120i Touring	9000	\$79990																																																																														
	2000	Chrysler Neon	15004	\$21995																																																																														
	1999	Chrysler Valiant	34000	\$31995																																																																														
	2000	Chrysler Voyager SE	500	\$49995																																																																														
	1999	Citroen Xsara SX	45095	\$22995																																																																														
NEW	2001	Daciaun Lotus SE		\$19495																																																																														

Figure B.8 Page Shots Using Traditional Approach

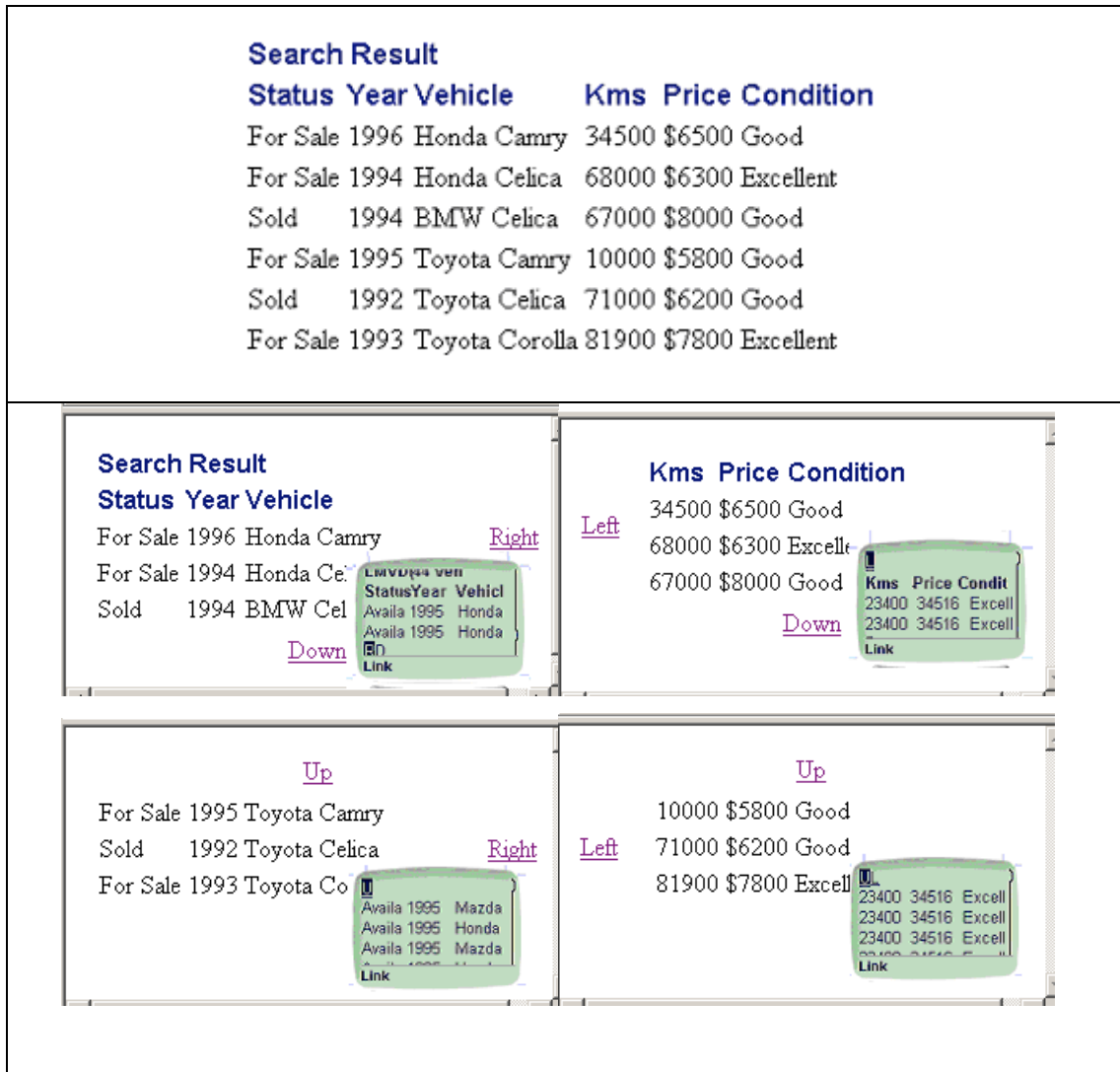


Figure B.9 Page Shots Using Adaptable Approach

Suppose we use the following window with size 160*80 pixels to display the above page. The above page was spitted to be four pages with several links provided on each page. User can view rest information by clicking these links. For example, if user clicks “Right” link shown on the first page, the page on right its side will show up and display the rest information.

The following interfaces show the car list on mobile device with window size (50pixels*8lines), and eight pages were generated to display full information, more links have been generated, user can view all the information by clicking the links "R"(Right), "L"(Left), "U"(Up), "D"(Down) on each window. The following only lists four pages.

Car Search

The following page displays a search form using adaptable approach with normal desktop IE browser. The following set of page dumps show the layout that displayed using mobile device. The above page was separated to be five windows with links linking each other.

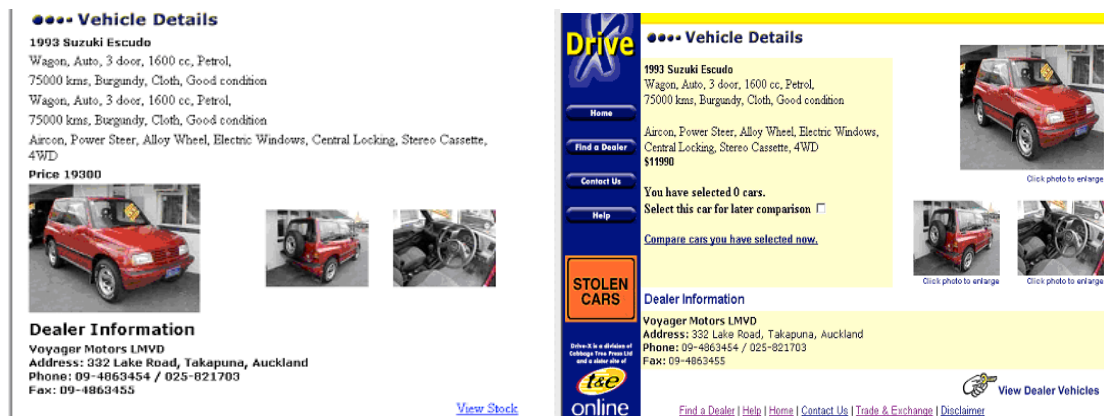


Figure B.10 Page Dump Using Two Approaches

4.2 Survey Questions

From the user interfaces developed for the car site, please answer the following questions:

Do you think the user interfaces developed using an adaptable approach are easy to use?

1. Very easy.
2. Easy.
3. Medium
4. Hard
5. Very hard

What do you think about the graphic layouts for user interfaces developed using an adaptable approach?

1. Very good
2. Good
3. Same as others
4. Bad
5. Very bad

Do you think if the user interfaces can provide sufficient functionalities?

Do you think it is easy to find information displayed on the user interfaces?

1. Very easy.
2. Easy.
3. Medium
4. Hard
5. Very hard

Do you think the screens load fast enough?

1. Very fast
2. Fast
3. Average
4. Slow
5. Very slow

For ease of navigation on small screen device, we have presented the information in small parts which the user works through progressively (see tutorial). Do you think this approach provides a satisfactory method of presenting data on a small screen device?

1. Very satisfactory
2. Satisfactory
3. No different to other methods
4. Unsatisfactory
5. Very unsatisfactory

Appendix C – Tag Library Description

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE taglib
  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
  "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>examples</shortname>
  <info>Simple example</info>

  <tag>
    <name>template</name>
    <tagclass>tagext.TemplateTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
      <name>bgcolor</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>title</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>type</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
  </tag>

  <tag>
    <name>group</name>
    <tagclass>tagext.GroupTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
      <name>width</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>height</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>align</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
      <name>bgcolor</name>
```

```
        <required>false</required>
        <rtexprvalue>>true</rtexprvalue>
    </attribute>

</tag>

<tag>
    <name>grouptr</name>
    <tagclass>tagext.GrouptrTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
        <name>cellheight</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>

<tag>
    <name>grouptd</name>
    <tagclass>tagext.GrouptdTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
        <name>cellwidth</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>align</name>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>colspan</name>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>

<tag>
    <name>table</name>
    <tagclass>tagext.TableTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
        <name>colspan</name>
        <required>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>

<tag>
    <name>tr</name>
    <tagclass>tagext.TrTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
        <name>colspan</name>
        <required>false</required>
```

```

        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>

<tag>
  <name>td</name>
  <tagclass>tagext.TdTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Simple Example</info>
  <attribute>
    <name>colspan</name>
    <required>>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>

<!--=====Form Tags=====-->

<tag>
  <name>form</name>
  <tagclass>tagext.FormTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Simple Example</info>
  <attribute>
    <name>name</name>
    <required>>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>method</name>
    <required>>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>action</name>
    <required>>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>

<tag>
  <name>textfield</name>
  <tagclass>tagext.TextFieldTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Simple Example</info>
  <attribute>
    <name>name</name>
    <required>>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>size</name>
    <required>>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>type</name>
    <required>>false</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>

```

```

    <attribute>
      <name>value</name>
      <required>>false</required>
      <rtexprvalue>>true</rtexprvalue>
    </attribute>
  </tag>

<tag>
  <name>button</name>

  <tagclass>tagext.ButtonTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Simple Example</info>
  <attribute>
    <name>name</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>value</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>type</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
</tag>

<tag>
  <name>select</name>
  <tagclass>tagext.SelectTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Simple Example</info>
  <attribute>
    <name>name</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
</tag>

<tag>
  <name>option</name>
  <tagclass>tagext.OptionTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Simple Example</info>
  <attribute>
    <name>name</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>value</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
</tag>

<!--=====Presentation Tags=====-->

```



```
<tag>
  <name>image</name>
  <tagclass>tagext.ImageTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Simple Example</info>
  <attribute>
    <name>height</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>alt</name>
    <required>>true</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>src</name>
    <required>>true</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>width</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>align</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
</tag>
```

```
<tag>
  <name>layout</name>
  <tagclass>tagext.LayoutTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Simple Example</info>
  <attribute>
    <name>face</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>color</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>size</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>align</name>
    <required>>false</required>
    <rtexprvalue>>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>bold</name>
```

```

        <required>>false</required>
        <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>text</name>
        <required>>false</required>
        <rtexprvalue>>true</rtexprvalue>
    </attribute>
</tag>

<tag>
    <name>break</name>
    <tagclass>tagext.BreakTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
</tag>

<tag>
    <name>link</name>
    <tagclass>tagext.LinkTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
        <name>direct</name>
        <required>>false</required>
        <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>param</name>
        <required>>false</required>
        <rtexprvalue>>true</rtexprvalue>
    </attribute>
</tag>

<tag>
    <name>label</name>
    <tagclass>tagext.LabelTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
        <name>text</name>
        <required>>true</required>
        <rtexprvalue>>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>allowcut</name>
        <required>>true</required>
        <rtexprvalue>>true</rtexprvalue>
    </attribute>
</tag>

<!--=====Other Function Tags=====-->
<tag>
    <name>param</name>
    <tagclass>tagext.ParamTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
        <name>name</name>
        <required>>true</required>

```

```
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>value</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>device</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>

<tag>
    <name>iterator</name>
    <tagclass>tagext.IteratorTag</tagclass>
    <teiclass>tagext.IteratorTagExtraInfo</teiclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
        <name>collection</name>
        <required>>false</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>

<tag>
    <name>user</name>
    <tagclass>tagext.UserTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
        <name>user</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>value</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>

<tag>
    <name>device</name>
    <tagclass>tagext.DeviceTag</tagclass>
    <bodycontent>JSP</bodycontent>
    <info>Simple Example</info>
    <attribute>
        <name>device</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
    <attribute>
        <name>value</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
</tag>
```

```
<tag>
  <name>task</name>
  <tagclass>tagext.TaskTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Simple Example</info>
  <attribute>
    <name>task</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
  <attribute>
    <name>value</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
  </attribute>
</tag>

</taglib>
```