

**SCALABLE USER INTERFACES FOR THE WEB**

by

Arman Danesh  
B.A., University of Toronto, 1989  
M.S., Boston University, 1991

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF  
THE REQUIREMENTS FOR THE DEGREE OF  
MASTER OF SCIENCE

In the School  
of  
Computing Science

© Arman Danesh, 2004

SIMON FRASER UNIVERSITY

September 2004

All rights reserved. This work may not be  
reproduced in whole or in part, by photocopy  
or other means, without permission of the author.

## APPROVAL

**Name:** Arman Danesh  
**Degree:** Master of Science (Computing Science)  
**Title of Thesis:** Scalable User Interfaces for the Web

**Examining Committee:**

**Chair: Dr. Jiangchuan Liu**  
Professor of Computing Science

---

**Dr. Stella Atkins**  
Senior Supervisor  
Professor of Computing Science

---

**Dr. Kori Inkpen**  
Supervisor  
Associate Professor of Computer Science  
Dalhousie University

---

**Dr. Eric Schenk**  
Supervisor  
Electronic Arts

---

**Dr. Robert D. Cameron**  
Examiner  
Professor of Computing Science

**Date Defended/Approved:**

*September 8, 2004*

# SIMON FRASER UNIVERSITY



## PARTIAL COPYRIGHT LICENCE

The author, whose copyright is declared on the title page of this work, has granted to Simon Fraser University the right to lend this thesis, project or extended essay to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users.

The author has further granted permission to Simon Fraser University to keep or make a digital copy for use in its circulating collection.

The author has further agreed that permission for multiple copying of this work for scholarly purposes may be granted by either the author or the Dean of Graduate Studies.

It is understood that copying or publication of this work for financial gain shall not be allowed without the author's written permission.

Permission for public performance, or limited permission for private scholarly use, of any multimedia materials forming part of this work, may have been granted by the author. This information may be found on the separately catalogued multimedia material and in the signed Partial Copyright Licence.

The original Partial Copyright Licence attesting to these terms, and signed by this author, may be found in the original bound copy of this work, retained in the Simon Fraser University Archive.

W. A. C. Bennett Library  
Simon Fraser University  
Burnaby, BC, Canada

## ABSTRACT

This thesis describes a new approach to developing and delivering user interfaces for Web applications. This approach, termed *Scalable User Interfaces* (SUI), is designed to allow a developer to create a single user interface definition for a Web application which can then be consumed, rendered and used by any device on the network. These devices can range from small displays such as mobile telephones to the full desktop-sized monitor displays used by personal computers.

The goal of Scalable User Interfaces is to allow a single specification to be deployed on all devices without the need for the developer to specify any device-specific vocabularies, transformations, hinting or style sheets such as previous work in automated rendering for mixed displays and work in platform-independent user interface specification.

Scalable User Interfaces provides a Flash-based implementation which highlights the utility of Flash as a tool for user interface design and research. Our work also illustrates the application of recursive rendering in laying out forms for various-sized displays.

## **ACKNOWLEDGEMENTS**

I owe a debt of gratitude to my supervisory committee for their patience during the long process of focussing my research and completing this thesis; Dr. Stella Atkins, Dr. Kori Inkpen and Dr. Eric Schenk all willingly worked with me through changes of topics until Scalable User Interfaces finally emerged. All my colleagues in the Edge Lab at Simon Fraser University also have my thanks for their support and ideas throughout my degree program. Finally, I thank my wife, Tahirih, for on-going support.

# TABLE OF CONTENTS

<b>Approval</b> .....	<b>ii</b>
<b>Abstract</b> .....	<b>iii</b>
<b>Acknowledgements</b> .....	<b>iv</b>
<b>Table of Contents</b> .....	<b>v</b>
<b>List of Tables and Figures</b> .....	<b>vii</b>
<b>Chapter One: The Web as Application Delivery Platform</b> .....	<b>1</b>
1.1 The Architecture of Web Applications .....	3
1.1.1 Connecting Clients and Servers .....	3
1.1.2 Moving to Web Services .....	5
1.2 The Trend to Small Devices .....	9
1.3 Build Once and Deploy Everywhere with Flash .....	10
<b>Chapter Two: The Dilemma of the User Interface</b> .....	<b>16</b>
2.1 Dynamic HTML .....	18
2.2 Java .....	22
2.3 Flash .....	24
2.4 Scalable Vector Graphics .....	28
2.5 Adobe Forms .....	29
2.6 The Shift to Small Devices.....	30
2.7 Approaches to Application Delivery in a Fragmented Client Space .....	33
2.7.1 Choosing Selected Browsers .....	33
2.7.2 Implementing Multiple Versions .....	34
<b>Chapter Three: Survey of Platform-Independent User Interfaces for Web Applications</b> .....	<b>37</b>
3.1 User Interface Management Systems and Automatic Rendering.....	37
3.2 Using Markup to Specify User Interfaces .....	44
3.2.1 Using XML to Simplify Application Delivery to Multiple Devices.....	45
3.2.2 Providing User Interface-Specific Markup with UIML .....	47
3.2.3 The Mozilla Experience: Using XUL .....	49
3.2.4 XForms: A Proposed Standard for Web Forms .....	51
3.2.5 Other Markup Languages .....	52
3.3 Addressing the Problem of Small Screens .....	53
3.3.1 Automatic Transformation of Content .....	53
3.3.2 Displaying and Navigating Large Data Sets on Small Displays.....	59
<b>Chapter Four: Scalable user interfaces</b> .....	<b>61</b>
4.1 Flash .....	61
4.1.1. Component Architecture.....	62
4.1.2 Built-In XML Parsing .....	63
4.1.3 Cross-Platform Compatibility.....	64

4.1.4 Flash on the Server: Macromedia Flex.....	65
4.2 SUIML: Markup for SUI .....	67
4.2.1 The Tag Set.....	68
4.2.2 Required Attributes .....	73
4.2.3 Tag-specific Attributes .....	73
4.2.4 Managing SOAP Services Callbacks .....	75
4.2.5 Current Implementation.....	78
4.3 Transformation Techniques for SUI.....	78
4.3.1 Rendering Forms.....	79
4.3.2 Displaying Result Sets.....	81
4.4 The SUI Rendering Engine API and Object Model .....	82
4.4.1 The Main Rendering Engine.....	83
4.4.2 Rendering Model API .....	85
4.4.3 Widget API.....	87
4.4.4 Modularity: Using Multiple Flash Movies .....	88
4.4.5 Flexibility and Configurability .....	90
4.5 Recursion.....	93
<b>Chapter Five: Sample Applications.....</b>	<b>96</b>
5.1 User Registration Form.....	96
5.2 Store Checkout Form .....	100
<b>Chapter Six: Discussion .....</b>	<b>107</b>
6.1 Flash as a User Interface Framework.....	107
6.2 Simple vs. Intelligent Algorithms.....	108
6.2.1 Problems with Small Displays .....	109
6.2.2 Problems with Large Displays .....	110
6.3 Client Performance .....	112
6.4 The Benefits and Drawbacks of Recursion.....	114
6.5 Standardization and Consistency.....	117
6.6 Implications for Developer and Designers.....	117
<b>Chapter Seven: Conclusions and Future Work.....</b>	<b>119</b>
7.1 Using Flash for Cross-Device Form Delivery.....	119
7.1.1 Implications for SVG and Other Platforms .....	120
7.2 Automated Form Rendering for Small Displays .....	120
7.3 Using Recursion and Simple Algorithms for Form Rendering.....	121
7.3.1 The Importance of Simple Algorithms.....	121
7.3.2 The Relevance of Recursion.....	122
7.4 Future Work .....	122
7.4.1 Test Alternate Rendering Algorithms .....	122
7.4.2 Implement the Complete Widget Set .....	123
7.4.3 Improve Efficiency of Algorithms for Small, CPU-Limited Devices .....	123
7.4.4 Conduct User Testing.....	124
<b>References.....</b>	<b>125</b>
<b>Appendix 1.....</b>	<b>135</b>

## LIST OF TABLES AND FIGURES

Figure 1	Multi-tiered architecture of modern Web applications .....	4
Figure 2	Typical Web Services Environment .....	8
Figure 3	The three layers of the SUI Architecture .....	11
Figure 4	Developing individual HTML files for each client .....	36
Figure 5	Typical UIMS Architectures .....	40
Figure 6	Transforming XML with XSLT for each client .....	46
Figure 7	Using UIML to build platform-independent user interfaces .....	48
Figure 8	Flash MX 2004's User Interface Components .....	63
Figure 9	A simple Flash movie containing a form on a handheld device .....	64
Figure 10	A simple Flash movie containing a form on Linux .....	65
Figure 11	Container-level elements in SUI .....	70
Figure 12	Widget-level elements in SUI .....	72
Figure 13	The stock application before and after the user enters data and clicks the button .....	77
Figure 14	The same form using the regular, card and minimal layout styles (from left to right) .....	80
Figure 15	SUI's Modular Architecture .....	83
Figure 16	A form for controlling settings .....	92
Figure 17	A mixture of regular and minimal layout .....	95
Figure 18	More space leads to regular layout throughout .....	95
Figure 19	Less space leads to minimal layout throughout .....	95
Figure 20a	A user-registration form in SUIML (640x480 pixels) .....	98
Figure 20b	A user-registration form in SUIML (320x240 pixels) .....	98
Figure 20c	A user-registration form in SUIML (240x320 pixels) .....	99
Figure 20d	A user-registration form in SUIML (150x150 pixels); here a card- based layout is used and the e-mail address card is displayed in the center .....	99
Figure 20e	A user-registration form in SUIML (90x90 pixels) .....	99
Figure 21a	A Store Checkout form in SUIML (640x480 pixels); image is scaled to fit on page. All fields fit relatively well and no scrolling is required .....	104
Figure 21b	A Store Checkout form in SUIML (320x240 pixels). At this size, a combination of widget styles appear in the different box elements .....	105
Figure 21c	A Store Checkout form in SUIML (240x320 pixels). All widgets are rendered in the minimal style .....	105
Figure 21d	A Store Checkout form in SUIML (150x150 pixels). At this size, minimal widgets are used but scrolling becomes necessary .....	106



Figure 21e	A Store Checkout form in SUIML (90x90 pixels). At extremely small sizes, there is no choice but to use minimal widgets .....	106
Figure 22	The store checkout form on a 90 x 90 pixel display .....	109
Figure 23	The shopping cart form on a large 1024 x 768 pixel display (rotated 90 degrees and scaled to 75% of actual size to fit page) .....	111
Figure 24	The store checkout form at 320 x 240 pixels .....	115
Figure 25	Operation of a minimal text widget .....	116
Table 1	Performance results for test applications .....	113

## **CHAPTER ONE: THE WEB AS APPLICATION DELIVERY PLATFORM**

As originally conceived by Tim Berners-Lee at CERN (Conseil Européen pour la Recherche Nucléaire) in the late 1980s and early 1990s, the World Wide Web was intended to be a network of linked, static hypertext documents to provide a more effective way for users at CERN to navigate complex sets of documents and information [14] [121].

By the mid-to-late 1990s, the Web had evolved beyond this concept to include increasing amounts of dynamic content as well as interactivity, generally provided through the use of CGI-BIN programming in Perl or C. By this time, researchers were noticing that the Web had applications beyond hypertext document linking into the areas of database interaction, even providing a platform for the delivery of applications to users. Aslam [8], for instance, noted in 1998 that one of the prominent uses of the Web at that time was for querying databases and in 2000, Labrinidis and Roussopoulos, in a study of CGI-BIN and mod\_perl programming [57] noted that the Web is in a trend towards data applications and that dynamic content had become the common denominator of most content delivery on the World Wide Web.

Recent years have seen the entrenchment of dynamic content on the Web with most major Web properties being driven by dynamic content through Web applications written in server-side application development languages such as Perl, PHP, ColdFusion, Java and ASP. In 2000, for example, it was estimated that more than one million Web sites were driven by PHP alone [41]. Even where sites are strictly concerned with delivering content, this content is now typically managed through content management

systems which store content in databases and provide simple tools for content developers to create, edit and manage the content which eventually is delivered to users dynamically from the database. Content management systems separate site structure, content and presentation/design into independent, separately managed layers and provide mechanisms for version control, pre-publication review and sign-off and more; to do this requires dynamic, data-driven tools rather than simple text-file editors used in manual HTML production [36] [21]. According to a study by the META Group, in 2002, 60 per cent of Global 2000 organizations owned Web content management systems with the number set to reach 95 per cent by 2004 [28].

In addition, as the World Wide Web has moved from static content to database queries to full-blown interactive applications, a parallel development has been the emergence of the Intranet concept in the mid 1990s and later [40]. In this paradigm, Internet technology such as HTTP and HTML are used within organizations to provide cross-department information sharing and provide easy delivery of corporate applications to users through Web technology [111].

In fact, the trend towards application delivery through the Web is so striking, several researchers have argued that the trend to dynamic content on the World Wide Web which emerged in the second half of the 1990s will ensure that the Web will emerge as the dominant platform for all application delivery [94] [89] [60]. As early as 1996, Rice et al. [94] noted that the Web would likely become the future platform for application delivery to users. They argue that the Web provides the benefit of decreased costs for distribution of software releases and upgrades as well as providing for portability across multiple platforms.

More recently, Rees [89] noted that computing was moving towards the use of the Web as the only user interface delivery platform and Lok et al. [60] pointed out that in modern computing, the personal computer is largely used as a terminal for running a client Web browser.

### ***1.1 The Architecture of Web Applications***

Currently, the Web client (typically an HTML browser) is used for delivery of the user interface. The HTML browser conducts transactions via HTTP with a Web server responsible for receiving and directing requests from multiple browsers. Requests are routed to an application server where the logic for the application exists and the application server processes these request against back-end data sources which store the data (typically databases) [31]. The result is a multi-tiered architecture for application delivery shown in Figure 1.

Typically, in a public Web application the developer has control of the technology deployed in the bottom three tiers but has no guarantees about the nature of the client software used to access the application. This means the developer needs to make decisions regarding the clients which will be supported by the application and account for these clients in the HTML-based user interface sent to the client.

#### **1.1.1 Connecting Clients and Servers**

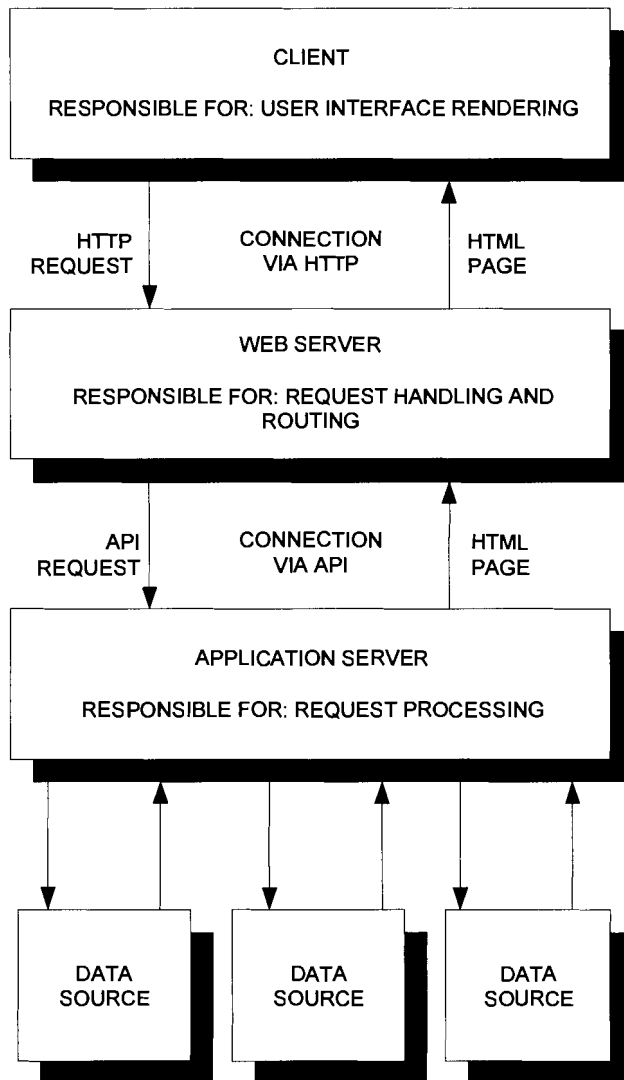
In practical terms, the client of these applications typically provides only two mechanisms for interacting with the back-end server layers:

- Links to request new documents or application pages from the server (the HTTP GET operation).

- Forms to submit data to the server for processing (the HTTP POST operation).

Until recently, even the most sophisticated Web applications which provide robust, user-friendly graphical user interface-style environments for users can still be distilled down to these two types of interactions with the server. If the design of the user interface is stripped away, the application will remain functional although it may not be terribly usable; the user can still, in principle, achieve the same results as a fully-designed application.

**Figure 1 Multi-tiered architecture of modern Web applications**



### **1.1.2 Moving to Web Services**

More recently, the interactions between the client and server have begun to change with the advent of XML-based Web services. The concept of XML-based Web services became a hot topic of discussion and development during 2002 as numerous Web application platforms, such as Macromedia ColdFusion MX [63] and Microsoft's .NET framework [75], began to offer integrated support for Web services.

The concept behind Web services is that instead of simply offering Web pages to client browsers delivered in HTML, Web application servers could also be used to expose functionality to remote invocation using Internet standards such as HTTP and XML to facilitate the transaction. Essentially, according to the Web Services Architecture working draft [26] maintained by the World Wide Web Consortium, Web services are a software system identified and accessed through a uniform resource indicator (similar to a URL) which exposes public interfaces and describes those interfaces with XML. The definitions of these interfaces can be discovered by other systems and interaction with the service is conducted using XML messages carried over Internet protocols.

In this model, for instance, a news organization which syndicates its headlines can allow other Web sites to invoke methods on their Web server which return XML-coded data containing the headlines and these other Web sites can process the data and then render as needed. Web services have been adopted in business-to-business environments where business partners can provide access to functionality from their internal systems through the Internet to remote business partners without being concerned about the compatibility between the system providing the service or the system consuming the service; in theory, with XML-based, standards-based Web services, the exact nature of the two systems involved in the transaction is irrelevant.

In addition, Web services are now being offered by many public Web sites. Amazon.com, for instance, offers Web services which allow other Web sites to search and display products from the Amazon.com web site directly in their sites and allow their visitors to add those items to an Amazon.com shopping cart [6]. Another interesting example is the DNA Databank of Japan which allows a variety of queries to be performed against its databases through Web services [34]. In addition, several on-line catalogues of public Web services have emerged; one prominent catalogue is XMethods [125]. XMethods currently lists more than 300 publicly accessible Web services including the Amazon.com and DNA Databank of Japan services; this list doesn't include internal Web services used within a corporation or only used between partner organizations.

In practical terms, modern Web services are offered using two XML-based languages:

- SOAP (Simple Object Access Protocol): SOAP is a communication protocol for sending messages between the client and server, typically over HTTP. Based on simple XML, it is the primary protocol used for carrying transactions between clients and servers offering Web services [90]. In many ways, SOAP is similar to RPC (Remote Procedure Calls) which is used for objects such as DCOM or CORBA to communicate over the network or Java-based technologies such as RMI. However, many other options for remote method invocation suffer from cross-platform compatibility problems, can suffer security problems and are not effective in the context of the public Internet where firewalls and proxy servers can block such traffic. SOAP over HTTP bypasses many of these limitations

providing cross-platform compatibility and bypassing firewall restrictions which would otherwise make offering remote services on the Internet a risky proposition.

- WSDL (Web Services Description Language): WSDL is an XML-based language used to describe the functionality of a Web service [91]. Publishers of Web services will also offer a WSDL description of the service, which details the location of the service and the methods the service exposes, including the necessary parameters which must be passed to the service and the type of results returned. It is WSDL that allows any system capable of consuming SOAP Web services to locate, decipher, and invoke methods from Web services running on any SOAP-capable server.

An important aspect of the Web services paradigm is that it moves Web application development into a more structured model where application logic is implemented in a Web service separately from the code which generates the HTML, or user interface, for the actual front-end of an application.

In this model, it is possible for many different Web front ends to exist which all use a single Web service which strictly implements application and data logic. This architecture is illustrated in Figure 2.

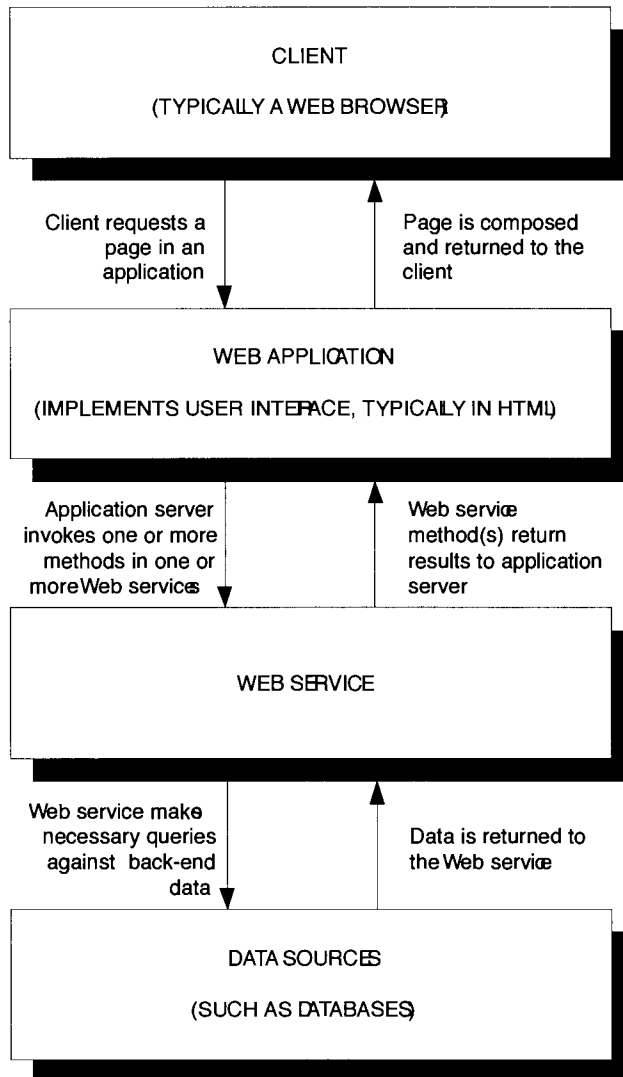
Numerous Web server application platforms available today can be used to develop and deploy SOAP/WSDL-based Web services, including:

- Macromedia ColdFusion MX [63]
- Microsoft .NET [75]
- Apache Axis [7]



- BEA WebLogic [12]
- Oracle Application Server [83]
- WebMethods Application Server [117]
- Macromedia JRun [64]

**Figure 2** Typical Web Services Environment



## ***1.2 The Trend to Small Devices***

The rapid growth of the World Wide Web has also resulted in a fragmentation of the client device space. Specifically, small, portable devices in the form of mobile telephones and personal digital assistants now have capabilities to access the Internet. Even a brief review of the mobile phone and personal digital assistant spaces shows that Internet access is becoming a key selling point with leading PDAs offering Internet browsing capabilities [45] [84] as do leading mobile phones [73] [96].

In fact, there is a growing consensus in the academic and industrial communities that the future of the Internet will be dominated by a broader diversity of client devices and that small devices will play a prominent role in this new Internet geography. According to Cerf [25], writing in 2001, Internet growth trends will likely lead to at least one billion interacting devices on the Internet by 2006 and most of this new growth will occur as the result of a notable increase in low-cost Internet devices in several categories:

1. Appliances (which arguably includes future personal digital assistants)
2. Sensors
3. Actuators
4. Communication devices (which includes mobile phones)

Similarly, Samulowitz et al. [98] suggest in their work on wireless and mobile Internet access that future environments are inevitably set to move towards mobile and wireless devices playing a more prominent role and Allaire [5] suggests that today's mix of new devices with new client technologies and Web services lies at the heart of a paradigm shift for Internet applications. Schilit et al. [100] also note that future evolution of the Internet will include a shift towards smaller devices.

These predictions are more than mere speculation. Evidence from Japan, which often leads the curve when it comes to adoption of new applications of technology, shows that Internet access through mobile phones has recently emerged as the most popular form of Internet access there [106].

### ***1.3 Build Once and Deploy Everywhere with Flash***

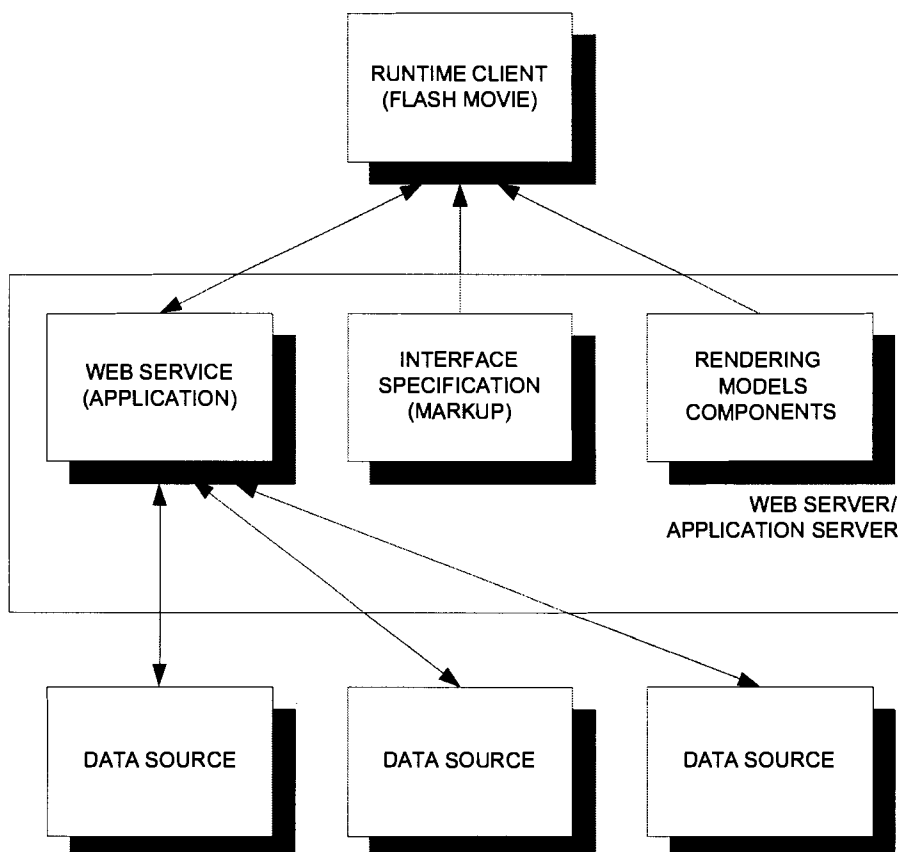
Scalable User Interfaces for the Web will attempt to address several issues which emerge in a review of the state of current Web applications and several of the lessons learned in work on user interface management system (UIMS) environments and in rendering Web content to small displays:

1. Richness of the widget set available: ideally, Web applications should have at their disposal rich widget sets.
2. Fragmentation of the client space: ideally, Web applications should be deployed to a single runtime environment that is consistent across devices.
3. Consistency of interface elements: ideally, on a given device, a user's experience of an interface should be consistent with experiences from other Web applications.
4. Separation of interface and application logic: user interface specification and back-end application logic should be kept cleanly separated.
5. Abstracting the interface definition: user interfaces for Web applications should not be specified in a way tied to the physics of desktop displays and then be transformed by an algorithm for display on small devices as this leaves small devices as second-class citizens among Web application client devices.

The goal is a write-once, deploy-everywhere development model for Web applications that eliminates the problems of cross-browser compatibility including the code management problem of developing multiple interfaces and the sacrifices in delivering interfaces to devices lacking browser features, and removes physical constraints of devices such as screen size and bandwidth limitations as a consideration in development of user interfaces for applications.

Once implemented, the development and delivery model for an interface using SUI should look like Figure 3. This architecture includes several important components:

**Figure 3**      **The three layers of the SUI Architecture**



- *Runtime client*: The runtime engine delivers the rendered interface to the user, handles user interaction and invokes Web service methods as needed.

The engine is implemented as a Flash application that is delivered across the network and runs in the user's Flash Player. The application was developed in Flash MX because it offers native XML support, integration with Web services and a rich library of interface widgets complemented by a component model for developing additional widgets programmatically. Flash MX eliminates the browser-compatibility problems of Dynamic HTML, does not suffer from the scalability problems common with Java applets and provides a robust rapid application development environment and object-oriented scripting language for application development.

- *Rendering model*: the rendering model is a Flash component which implements a rendering or transformation algorithm for rendering a platform-independent user interface on a target device accounting for screen size and other device limitations and capabilities. The runtime client loads the rendering model component before loading the user interface specification. By separating the rendering model into a component, it is possible to experiment with different models by building alternate components without altering the runtime engine.
- *User interface specification*: the user interface specification consists of a platform-independent definition of a Web data application's interface written in an XML-based markup language. The runtime engine loads this file after the rendering model component is loaded. The rendering model component processes the user interface specification and the results are

used by the runtime engine to display and manage the user interface experience.

- *Web service*: The user interface application specification includes specification of how the user interface interacts with methods in a back-end Web service. The runtime engine can interact with any standard SOAP/WSDL Web service implemented in Macromedia ColdFusion MX, on a Java J2EE server or through Microsoft's .NET framework. For the purposes of sample applications developed for testing SUI, the Web services were implemented using ColdFusion MX which provides the simplest environment for developing and deploying SOAP-based Web services.

In many ways, this model reflects earlier external control UIMS implementations in that the interface layer controls the application. In SUI, the runtime client, once it renders the interface invokes Web service methods as subroutines in much the same way as was done in external control UIMS environments of the past. However, unlike systems of the past which were closed environments, SOAP/WSDL provides an open, standard mechanism for implementing these subroutine-like applications so that the Web service methods are equally invocable from Web applications deployed using other technologies including Java, Visual Basic .NET, Visual C# and other popular Web application development languages. This means that SUI-based applications are not tied to the SUI environment, because the back-end which handles all the application and data manipulation logic is implemented using a standard open technology.

Similarly, because an XML-based markup language is used for user interface specification, if developers later want to migrate applications off of SUI back to

traditional Web delivery mechanism or towards future techniques, XSLT can be used to transform the user interface specifications into any other standard markup for delivery to non-SUI client browser software.

This project will target the specific domain of Web data applications. That is, those applications which are common today for obtaining data from the user, processing that data against a database and returning data to the user for display. Common applications which fall into this domain are:

- Shopping carts and checkout applications
- Search Engines
- Registration Forms

As most of these applications are moving towards SOAP and WSDL-based Web services models, Scalable User Interfaces for the Web will implement a generic client tool for deploying applications which communicate with Web services back end systems.

Scalable User Interfaces extend current work in several areas:

1. *Delivery of Web applications to multiple clients:* instead of taking an automatic rendering approach to re-render desktop content for small displays, Scalable User Interfaces dynamically renders a platform-independent user interface definition on all displays.
2. *Markup language for user interfaces:* SUI implements a derivative of the XUL markup language for specifying the components of Web application interfaces. Several key tags are adopted from the XML User Interface Markup Language (XUL) from the Mozilla and then SUI-specific attributes have been defined (XUL is discussed in Section 3.2.3). The

language developed provides mechanisms for developers to specify how to respond to user actions and how to tie the interface to a backend Web service.

3. *Transformation techniques*: although SUI does not transform desktop content to small devices, transformation techniques need to be applied to an interface definition in deciding how to render the interface.



## **CHAPTER TWO: THE DILEMMA OF THE USER INTERFACE**

Even though some would suggest that the evolution and adoption of the Web as an application delivery platform argues that the Web browser is destined to be the dominant user interface tool, the growth of the Web has also been plagued by a range of challenges which limit its ability to be a truly ubiquitous platform for interactive application design and delivery.

Historically, the largest problem affecting the Web's utility as an application delivery platform has been the limited interactive capabilities and widget options of HTML [97] [44] [60]. HTML-based Web applications have also lacked support for event-based programming and cannot maintain state [60]. In addition, these applications suffer from bandwidth and server resource inefficiencies in that each user interaction with the application requires the complete delivery of a new page with a complete user interface. This consumes bandwidth to redeliver the user interface and requires the server to reconstruct the user interface for delivery to the user [44].

Most notably, the Web in its current form as a mix of browsers, plug-ins and server software has limited the ability of developers to create robust, interactive applications which function well across the breadth of the fragmented Web client space. As Rees et al., point out, there is a need for a universal Web browser [89]. However, one does not exist today.

As the Web has evolved, the limitations of plain HTML-based applications have been addressed by the use of several technologies: Dynamic HTML, Java Applets, and

ActiveX controls [44]. These have helped to address the problem of limited widgets and, to a limited degree, the ability to trap user events. However, none of these approaches are universally favoured nor can they be considered robust and complete. ActiveX Controls only run in Microsoft Internet Explorer on Windows systems; Dynamic HTML and Java suffer from their own limitations which will be discussed later in Section 2.1.

The limitations in traditional HTML-based applications along with the lack of a single, universally-accepted technology for the delivery of interactive applications create barriers to the delivery of interactive applications through the Web, especially where the goal is to provide a rich, interactive user experience.

Macromedia has put forward the notion of rich Internet applications [67] [66] [44] [61]. The motivation for rich Internet applications is that, historically, Web-based applications have provided easy delivery of applications and user interfaces to the users but have been limited in terms of user interactivity, content organization and design, and flexible data manipulation capabilities [1] [4] [97]. Desktop software, by contrast excels in these areas but is harder to deliver to users and keep up-to-date [1] [5]. Macromedia argues that rich Internet applications will offer the best of both worlds: easy delivery to the user while providing robust user interactivity, content organization and data manipulation capabilities.

This notion of providing desktop-like functionality and interfaces using the Web as the delivery mechanism has also been voiced by researchers. Lok et al. [60], for instance noted that the move to network-centric computing and the browser as a terminal means that it is necessary to develop new toolkits for the creation and delivery of user interfaces on the Web. Rees [89], as well, acknowledges that the move to Web-based user

interfaces brings with it new HCI requirements and requires new research and training of HCI practitioners in developing effective applications for Web delivery.

The Web as it is currently used provides several approaches to delivering rich Internet applications:

- *Dynamic HTML*: Dynamic HTML is the combination of HTML with an object model accessible in the client through JavaScript which allows programmatic manipulation of page elements in the client display. Creating Dynamic HTML applications requires the use of HTML, JavaScript and Cascading Style Sheets [59].
- *Java*: Java applets can be embedded in Web pages to provide GUI-style applications or application elements which are network enabled and can work with back-end data on the Web server [107].
- *Flash*: Historically, Flash has been viewed as a tool for delivering multimedia content such as animations for embedding in Web pages. In reality, though, Flash provides a platform for the creation of application interfaces or interface widgets for use in larger HTML application interfaces. At the current time, when Flash is adopted for application delivery, it is typically limited to individual interface widgets such as dynamic, interactive menus [62].

## ***2.1 Dynamic HTML***

While no hard statistics are available which indicate how many sites choose Dynamic HTML instead of Java or Flash for interactive user interface development for Web applications, a quick survey of major Web sites clearly indicates that Dynamic

HTML is a widely adopted approach to solving the problem of delivering rich Internet applications to users.

For example, the home page of *traveolcity.com* provides an example of a rich Internet application deployed with Dynamic HTML. The home page provides a small form for searching for flights, hotel rooms, and car rentals. It provides numerous dynamic features representative of the types of rich Internet applications currently found on the Web, including:

- A tabbed display which allows users to switch between panels in the client using CSS and JavaScript.
- Dynamic forms which adjust and change based on the user's input. For instance, if a user chooses the "Flexible Dates" option when searching for a flight, the form changes from providing specific departure and return dates to just allowing users to select a range of months.

The largest advantage of Dynamic HTML is its relative simplicity when compared with Java. Dynamic HTML is an amalgam of HTML, an object model for Web pages and browsers, JavaScript and Cascading Style Sheets. When combined, these standards offer the potential for the creation of complete, interactive user interfaces with a simple development cycle. Building Dynamic HTML applications does not require complex frameworks or the skills of object-oriented applications developers [5].

A casual survey of major Web sites shows that the use of Dynamic HTML is the most common approach to developing rich Internet applications; this has brought to the fore a critical concern in the present Web environment: browser compatibility. The support for the HTML, JavaScript, object model and style sheets underlying Dynamic

HTML varies widely across the range of Web browsers currently in use. This includes internal variation between versions of a single browser as well as variations between different browsers [114].

This has led to development processes which are not as simple as promised in theory with Dynamic HTML. Typically, a Web application interface delivered with Dynamic HTML has to account for multiple browsers and versions with subtle differences between the implementations and support for Dynamic HTML standards.

In addition, these applications have to account for, and handle, cases where users are using browsers which don't support Dynamic HTML or support only very small parts of the Dynamic HTML standard. These browsers include older browsers which are still in use in some organizations and some parts of the world as well as new browsers emerging in the wireless space where mobile devices such as personal digital assistants and mobile phones have the ability to access the Web but often use browsers which are limited in their support for the complete HTML standard and Dynamic HTML.

The result of this mix of target clients is that a robust Web application which should be universally accessible typically needs to account for the following cases:

1. Feature-limited HTML clients (such as HTML browsers in handheld devices)
2. HTML clients with limited or no Dynamic HTML support (such as older Web browsers)
3. Dynamic HTML clients; these clients are then further broken down into sub-cases by browser type and version, including:
  - a) Microsoft Internet Explorer (versions 4.x, 5.x, 6.x)

- b) Netscape Navigator (versions 4.x, 6.x, 7.x)
- c) Mozilla (version 1.x)
- d) Opera (version 6.x, 7.x)

Further fragmentation occurs by platform. For instance, Internet Explorer is available in various versions for Windows, Mac OS and Mac OS X. Even for the same numerical version of Internet Explorer, implementation and support for Dynamic HTML varies. For instance, Dynamic HTML support between Internet Explorer 5 for Windows and Internet Explorer 5 for Mac OS X is different. Similarly, Mozilla is available on multiple platforms as is Opera.

For example, the Cascading Style Sheets component of Dynamic HTML exhibits severe browser fragmentation. In the early development of Web browsers, such as the 4.x versions of Netscape and Internet Explorer, vendors often opted to ignore established standards for CSS in favour of custom extensions of CSS and non-standard interpretation and rendering of CSS. More recently, all major browser vendors have moved in the direction of compliant implementation of CSS specifications but no single browser yet can claim a perfect, complete implementation of the current CSS specification. The result is variation in which style attributes are supported and how they are supported across different browsers. For instance, the white-space attribute which controls the behaviour of white space in documents, is implemented in Internet Explorer 5 for the Macintosh but not Internet Explorer 5 for Windows; similarly, the float attribute which allows for the creation of floating layers, exhibits significant differences in implementation between the modern versions of Netscape and Internet Explorer for Windows and is not even fully implemented on Internet Explorer 5 for Windows or earlier versions.

In addition, this list doesn't include newer or less well-known browsers such as Safari (Apple's attempt to introduce its own browser for Mac OS X) or Konqueror, the default browser in the KDE desktop environment under Linux and various Unix versions.

This diversity of possible clients may mean that a developer has to manage more than a dozen versions of the client implementation of a Web application if the goal is universal application delivery through the Web.

## ***2.2 Java***

Java was originally conceived as a true cross-platform application development language both for developing stand-alone desktop software as well as for developing applets for embedding in Web applications [107]. Java is supported by major Web browsers which can run embedded Java applets.

Java applets can provide robust user interfaces using Swing and other user interface libraries available to Java programmers and, in theory, can run on any platform identically.

However, Java suffered several drawbacks in its history which have stopped its uptake as the platform of choice for delivering application interfaces through the Web:

- Early implementations of the Java Virtual Machine, especially in embedded use in Web browsers were slow and unstable; this meant it wasn't practical to deploy Java applets in production Web applications. Early anecdotal evidence includes experiences with Web browsers crashing when loading pages containing Java applets, particularly on non-Windows operating systems. Even today, anecdotal evidence points to users complaining that Java applets suffer from poor performance.

- The battle between Microsoft and Sun over Java led to a Microsoft implementation of the Java Virtual Machine which wasn't 100% compatible with the Java standard maintained by Sun. Through the courts, Sun has successfully sought a partial resolution of the problem but there are still numerous users out there with these non-standard Java Virtual Machines on their systems. The end result of this is that, in theory, it is possible to build a Java applet which cannot run on every system [109].
- Java suffers from some scalability problems. While there are Java virtual machines for even small footprint devices such as handheld computers and mobile telephones, the Java Virtual Machines on these smaller devices are often feature-limited and don't implement the complete Java specification. This makes it hard to implement a robust, feature-rich, network-capable Java-based Web application if it is intended to run on any device from small handhelds to full-scale personal computers. In particular, Sun has released the Java 2 Micro Edition (J2ME) specification designed for handheld devices which is not a complete Java 2 Standard Edition (J2SE) implementation. This means a truly portable application must target only features common to J2SE and J2ME [108]
- Java suffers from complexity. To develop anything more than the most simplistic application requires a sound understanding of object-oriented programming and formal programming training. Dynamic HTML and Flash, on the other hand, are sufficiently accessible that many non-programmers master the skills needed to produce quite sophisticated applications [5].



## 2.3 Flash

Historically, Flash has been viewed strictly as an animation tool for presenting fancy special effects and introductory pages, known as *skip intros*, on Web sites [71]. While the earliest versions of Flash were limited in this way, as the Flash environment matured it became a robust tool for application development.

A quick review of the Macromedia Showcase [65], a section of the Macromedia site highlighting innovative uses of Macromedia technology, provides numerous examples of applications implemented in Flash, including:

1. Quakeroatmeal.com offers an interactive food planner for children to help them learn proper nutrition [88].
2. Wizeguides.com offers an interactive map of Boston's transportation system and attractions [120].
3. Manchester Airport offers an application for viewing arrivals and departures information [68].

As a technology, Flash includes several components:

4. *The Flash development environment*. This is a complete IDE for creating Flash files, known as *movies*. This IDE provides a complete graphical environment for creating user interfaces and application design as well as a complete, object-oriented language known as ActionScript which is a variant of ECMAScript (JavaScript is, likewise, a variant of ECMAScript [35]) [4].
5. *The Flash binary file format*. This file format is an open format in that the specification is available and products other than the Flash development

environment from Macromedia are available for creating Flash movie files. Most of these applications are designed to make it easier for non-technical users to create attractive presentations and animations in Flash movies rather than providing robust application development capabilities. For example, Swish is a product for creating Flash-based animations without any programming using an interface simpler to learn than the standard Flash development environment [110].

6. *The Flash player.* The player is available freely from Macromedia and is produced for almost every significant end-user computing platform currently available. The Flash player is available on the following platforms:

- a) Microsoft Windows 95 through XP
- b) Mac OS X
- c) Mac OS 9
- d) Linux
- e) Pocket PC
- f) OS/2
- g) Solaris
- h) HP-UX
- i) SGI Irix
- j) Selected mobile phones

The major difference between the Flash player and implementations of the Java Virtual Machine is that a given version of the Flash player will be functionally equivalent across all platforms. For instance, Flash Player 7, once implemented on all platforms, will behave identically when playing a specific Flash movie on all those platforms (accounting for functional differences in the devices such as screen size and input devices) [4]. By contrast, Java has multiple platforms such as J2SE and J2ME [107]; in addition, different vendors implement the Java virtual machine and there have been cases of implementations being incompatible with the standard as was the case with Microsoft [109].

This means that Flash succeeds, in some ways, where Java has failed: to provide an environment in which applications can be built which will genuinely run on any device. The critical difference is that Java is an open standard; there are open development tools for Java and while the specification is controlled by Sun the actual development environment and virtual machine may be implemented by anyone. By contrast, Flash is a Macromedia product and is tightly controlled by Macromedia. To some, this is a limiting factor to Flash because it means that if Macromedia were to choose to stop developing Flash as a product or Macromedia were to cease operation, Flash could quickly lose its cachet as a popular Internet development tool.

Nonetheless, Flash enjoys a ubiquity currently lacking for almost any other technology for developing and delivering the client side of Web applications. According to figures published by Macromedia, the Flash player is installed on the systems of more than 97% of Web users [44]. That means 97% of users with Web browsers on their computers have the Flash player installed. This is more pervasive as a target platform for application development than Dynamic HTML or Java.

While a few more users have Web browsers installed than the Flash player installed, these browsers represent a diverse mix of versions and no one browser and version enjoys the penetration of the Flash player [114]. Even Microsoft Internet Explorer, which collectively enjoys in excess of 90% of the Web browser market, suffers from notable version fragmentation.

A critical advantage to Flash is that upgrading the Flash player is typically a simple, automated process which initiates the first time the user attempts to view a Web site containing a Flash movie requiring a newer version of the player; the process is generally non-intrusive and fast. Upgrading a Web browser on the other hand typically requires large file downloads and a complete installation process and possibly a system reboot. For example, to install the current version of Netscape for Mac OS X requires a 58 MB disk image while the current version of the Flash Player for the same platform is 1.1 MB.

For this reason, users may be less likely to upgrade a browser just to gain access to a specific Web application than they would be to upgrade the Flash player to the latest version; current browser use statistics suggest this is the case since as many as 20% of Web users still use Internet Explorer 4.x or 5.x versions even though 6.x has been available for more than a year. A March 2004 survey of Flash version use by Macromedia shows that 93% of Flash users have at least Flash version 6 on their systems. In the first year after its release, penetration exceeded 86% and 15 months after release had exceeded 90%. Similar statistics are not yet available for the current version, version 7, as it has not yet been available for a full year.

In addition, newer versions of the Flash player are backwards compatible with Flash movies built for earlier versions; with Dynamic HTML, there is no guarantee that

code developed for Internet Explorer 5 will behave and render in precisely the same fashion when viewed with Internet Explorer 6.

Similarly, differences in versions of the Java virtual machine, compounded with the fact some users do not have Java virtual machines installed on their system, means that Java enjoys less pervasive use than the Flash player as well. For instance, Microsoft no longer ships Java in Windows and leaves it up to the user to obtain Java from another source if they require it [74].

Nonetheless, Flash suffers a problem of perception. For most Web developers and users, Flash is still perceived as an animation tool and not as a tool for application development.

## ***2.4 Scalable Vector Graphics***

Scalable Vector Graphics (SVG) is an emerging standard developed by the World Wide Web Consortium (W3C). SVG is designed as a platform two-dimensional vector graphics, using an XML-based file format and scripting built on ECMAScript with support for animation [122].

In many ways, SVG promises similar features and capabilities as Flash:

- SVG allows for implementations that run on resource-limited devices such as mobile phones and handheld computers.
- SVG provides for consistent visual rendering across platforms.
- SVG's support for scripting allows for the creation of interactive user interfaces and applications.

However, SVG does not presently appear to enjoy the ubiquity of Flash. Although no statistics are available regarding the installed base of SVG viewers, several factors suggest SVG is not yet ready to be used as a common application platform:

- Informal consideration of a range of Web users shows that users with Flash installed generally do not have an SVG viewer installed.
- There is no single, dominant SVG viewer: instead there are a range of experimental, development and production viewers for various platforms [123].
- Leading Web development and graphics tools do not all yet support SVG.

In the future, though, it does appear that SVG could grow into a platform for deploying platform-independent, graphically-rich user interfaces and applications for the Web but at the present time Flash is a more widely deployed, accepted platform for achieving these sorts of applications.

## ***2.5 Adobe Forms***

The Adobe Form Server is Adobe's solution to the problem of delivery of interactive forms to users on multiple platforms [126]. An enterprise-scale server application, the Form Server allows form designers to create a single form definition using an XML markup language. The server, in turn, can transform the form into a PDF-based form for delivery to Acrobat Reader, into a full Dynamic HTML application for delivery to a full-featured desktop Web browser, or into a limited HTML document for feature-limited devices such as handheld devices. The Form Server handles data collection and validation and supports back-end communication with Web services.

The strategy used by Form Server is to transform the single specification into different output based on the feature set of the target client (Adobe PDF, Dynamic HTML or plain HTML). This is a proprietary platform with the associated licensing fees required to deploy the technology.

## ***2.6 The Shift to Small Devices***

The shift to small devices compounds the user interface and browser compatibility problems just described.

In terms of user interfaces, developers now face additional limitations. Schilit et al. [100], outlines several key limitations of small devices:

- *Limited input capabilities.* For instance, many personal digital assistants use pen-based input (for example, the Palm Zire [84]) or newer thumb-based keyboards (for example, the Handspring Treo [45] and the Research In Motion Blackberry [93]). Mobile phones typically use numeric keypads for all input.
- *Lower bandwidth.* As an example, GPRS, a popular digital packet service available for mobile phones, may only offer speeds comparable to a 56 kbps dial-up modem connection in some situations [22].
- *Slower processors.* While the processors in some PDAs have rapidly improved in performance in recent years (some processors now run in the range of 400 MHz) [48], this is still far outpaced by the 2 GHz and faster processors in new personal computers available today. Mobile phones tend to lag even further behind [86].

- *Small memories.* Many PDAs could provide memory in sizes similar to average desktops but don't because memory in PDAs is typically flash memory which is more expensive than conventional random-access memory [30]. In addition, PDAs and mobile phones often lack any type of storage alternative comparable to a computer's hard disk which provides large amounts of additional storage which can be used for swap or persistent storage of data outside active operating memory (some devices offer support for memory cards such as Compact Flash but this is not yet ubiquitous on handheld devices; most notably, mobile phones usually do not offer this feature usually because of size and voltage requirements although this is beginning to change with the introduction of smaller Flash cards [102]).
- *Small displays:* In terms of delivering effective application interfaces to small mobile devices, this limitation is significant. According to Trevor et al. [112] screens are on a trend towards smaller sizes which necessitates rethinking the delivery of Web content to handheld devices. While some devices such as the larger Pocket PC PDA devices have screen resolutions (as much 240 by 320 pixels) which may accommodate simple scaling of Web content to fit [100], in most cases screen sizes on mobile device have only a fraction of that resolution. Schilit et al., provides an overview of common handheld screen sizes on mobile phones and PDAs in 2001 with screen sizes ranging from as little as 96 by 32 pixels up to 160 by 160 pixels on Palm PDA devices. Pocket PC-based PDAs were not included in



this overview as the study was focused strictly on the smallest devices with the Palm offering the largest display in that category.

This combination of limitations raises fundamental issues about how to deliver Web application interfaces to small form factors since it is unlikely that the typical Web page, as currently designed for desktop displays averaging 800 x 600 pixels, will render easily on devices of less than 100 x 100 pixels.

In addition to the physical limitations of handheld devices, introducing these mobile units into the client mix for Web applications further accentuates the browser-compatibility issues described earlier in the context of traditional desktop operating system Web browsers.

Typically, mobile devices do not provide support for HTML, or if they do, they offer extremely limited support for HTML. Schilit et al. [100], noted that Web browsers for handheld devices, which do implement HTML for their browsers, only implement a subset of HTML. In addition to HTML, many handheld devices offer browsers based on other markup languages, including:

- HDML (Handheld Device Markup Language) [2]
- WML (Wireless Markup Language) [92] [2]: WML is a descendant of HDML
- CHTML (Compact HTML) [54]: CHTML is an older proposed subset of HTML for phones, PDAs and other small mobile devices.

The modern mobile Internet space is largely dominated by WML with most phones offering Internet access with integrated WML browsers.

The implication of this is that as mobile devices are increasingly viewed as valid target clients for Web applications, developers must now not only account for variations in support for HTML and Dynamic HTML but now must also produce versions of their applications in WML for the mobile users.

This has serious consequences. Most notably, WML is even more limited than HTML in its ability to specify and deliver sophisticated user interfaces of any sort [92].

## ***2.7 Approaches to Application Delivery in a Fragmented Client Space***

The result of this varied client technology is that developers have two main choices for addressing browser compatibility in developing their applications:

1. Choose one or a select few browsers and restrict users to these clients. In a closed corporate environment this is feasible but for applications targeted at public consumption or for use by an unregulated user population, this can severely restrict the user base for the application.
2. Make the application accessible to all potential users by creating multiple variations of the underlying Dynamic HTML code driving the interface. Typically, this may mean creating code variations for anywhere from a handful to more than a dozen potential clients or developing a single monolithic Dynamic HTML application which handles exception cases for all these clients.

### **2.7.1 Choosing Selected Browsers**

Some Web applications are designed to support selected browsers. Some sites and applications on the Internet also explicitly indicate they only work with selected clients.

A classic example of this is Microsoft Outlook Web Access [37]; this application is a

Web front-end to a Microsoft Exchange server and allows users to access their folders on the server via the Web in an interface that is considered comfortable for Outlook users. This Web application requires the use of recent versions of Microsoft Internet Explorer.

In the public Internet space, sites may require a specific browser or a specific operating system in order to work. This is not a significant problem in the typical Intranet where an organization's users share a common computing environment. However, on the Internet, with its diversity of client browsers, devices and operating systems, developers seeking broad access to their applications cannot choose to exclude browsers [114].

### **2.7.2 Implementing Multiple Versions**

Implementing multiple versions of an application interface to support the possible range of client devices and software is a daunting task but one that is often adopted. Most major Web sites take steps to ensure the availability of their content on multiple browsers, and numerous Dynamic HTML libraries are available to assist developers in their attempts to develop interactive applications that are functional on a wide range of browsers [105] [29].

However, the broader the range of browsers supported, the harder this task becomes. Inevitably, it evolves to the point where distinct versions of the application's user interface are implemented for each target browser or group of browsers in order to account for the individual feature sets of the targets.

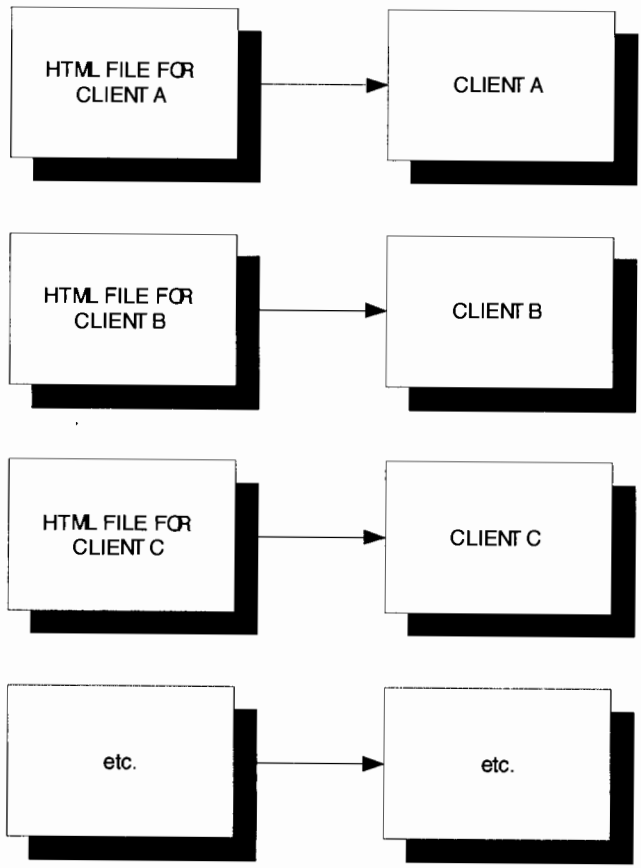
The end result, however, is a development challenge and code management problem. Maintaining multiple versions of the user interface code is fraught with problems including debugging challenges and the likelihood of increasing divergence in

the behaviour of the various versions of the interface as the application ages. This implies an application design which looks like Figure 4.

Including handheld devices makes the problem significantly more difficult since numerous researchers are indicating that the user interface and navigation paradigms for small screen displays may have to be fundamentally different than on desktop computer browsers. This would mean not only developing slight variations of an interface for different HTML-based browsers but also developing fundamentally different user interface paradigms for handheld devices.

For example, Trevor et al. [112] posit that small screens mean that the Web browsing strategy offered to users must be fundamentally different than it is on the desktop. At the same time, numerous studies suggest that the way in which information is presented must adopt new metaphors on small screen displays [72] [100] [70] [23].

**Figure 4**      **Developing individual HTML files for each client**



## **CHAPTER THREE: SURVEY OF PLATFORM-INDEPENDENT USER INTERFACES FOR WEB APPLICATIONS**

### ***3.1 User Interface Management Systems and Automatic Rendering***

Since the mid-1980s, developers and researchers have been seeking mechanisms to simplify and automate the development of user interfaces and to generate multiple interfaces for a single application [47]. This led to the emergence of the field of user interface management systems (UIMS) which attempted to provide several advantages over traditional manual development of user interfaces by application developers:

- Decreased effort in constructing user interfaces [47]
- Reduction in the programming skill threshold required to develop interfaces [47]
- Assurance of consistency across applications [47]
- Separate application from its interface [118]
- Easy implementation of multiple interfaces to a single application [47]

In many ways, developers of early UIMS platforms were trying to solve the same problems current developers of Web applications are trying to solve. Web delivery of applications is designed to provide several advantages which parallel the five advantages of the UIMS concept stated above:

1. Declarative language such as HTML and WML used to specify Web interfaces require less effort than procedural languages when specifying interfaces [1].

2. Declarative languages are easier to master and use than imperative languages traditionally used to develop user interfaces and, accordingly, allow programmers with less skill to develop user interfaces.
3. HTML provides an assurance of consistency across applications (in other words, the <SELECT> tag will render the same way for all Web applications rendered in the same browser).
4. The current trend towards Web services provides for a clean separation of presentation and application logic.
5. The clean separation of applications from their interfaces makes it easier to deploy multiple interfaces. As discussed later in this document, the current trend towards the use of XML on the Web allows for the easy transformation of content into multiple forms using style sheet transformations.

It is important to note the difference between UIMS approaches and user interface toolkits. Toolkits provide a set of standard libraries which can be used to implement user interfaces; by encapsulating the details of user interface widget implementation into library methods, the developer can be assured of the consistency of behaviour within an application and can avoid some of the complexities of user interface implementation. Weicha et al. [118], outline the difference between UIMS systems and toolkits: toolkits separate style of presentation from the application while UIMS platforms separate the user interface from the application. The result is that toolkits make it difficult to transfer an application to another development environment, cannot be used to state global policies throughout an application, and cannot be used to make global changes to an

interface [87]. In addition, toolkits don't allow the creation of new interaction techniques without programming and don't help designers decide when to use a specific type of user interface element or interaction [118]. UIMS environments, on the other hand, allow the development of multiple interfaces and can be used to assure consistency within and between applications [118].

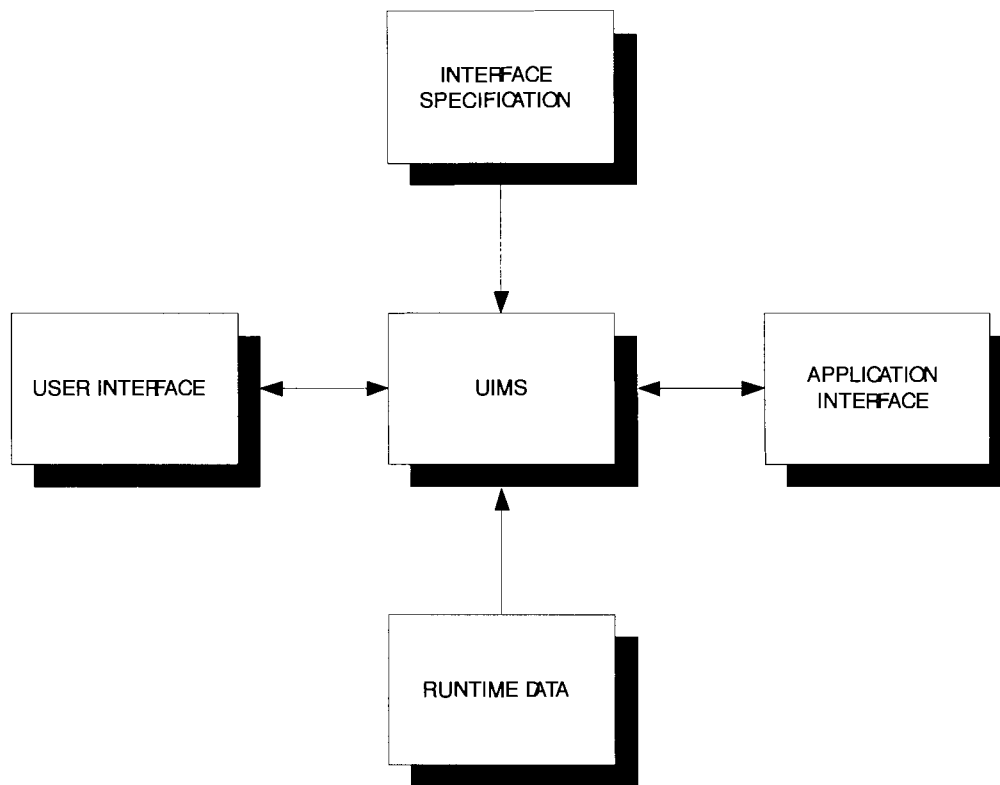
In generalized terms, the architecture of a UIMS is easy to understand. This architecture consists of three main components illustrated in Figure 5 [47]:

1. A user interface: this provides the programmatic interface to the actual UI experienced by the user. Effectively this is an API allowing the UIMS to manipulate the user interface delivered to the user.
2. The UIMS Platform: this accesses runtime data and an interface definition to tie together the user interface and the application interface.
3. An application interface: provides a programmatic interface to the application logic. This is an API allowing the UIMS to interact with application logic.

Essentially, the role of the UIMS is to bind together a user interface and an application. The application exposes a set of standard methods (its interface) which the UIMS can use; similarly, the UIMS creates and interacts with the UI itself through a set of standard methods. In order to manage this interaction between the user interface layer and the application logic layer, the UIMS relies on a specification of the interfaces, or APIs, for the user interface layer and the application logic.



**Figure 5** Typical UIMS Architectures



In practice, there are three types of UIMS implementations according to Hayes et al. [47]: internal control UIMS architectures, external control architectures and mixed control implementations.

Internal control architecture UIMS systems are built around the premise that the application controls the operation. That is, the application, through the UIMS, generates the interface. Many of the early UIMS systems of the mid-1980s were of the internal control variety [47]. While this is the most common form of UIMS [47], it presents a dilemma in that some of the purported advantages of UIMS design are lost. In particular, the application is tied into the interface in that the application code must invoke the necessary objects in the UIMS to generate the interface. This means the application code no longer stands on its own. At the same time, this integration of user interface logic into the application logic makes it hard to use an internal control UIMS platform to deploy

multiple interfaces for the same application since it would require changes to the application itself.

External control applications reverse the situation. In this model, the application is treated as a set of subroutines which are invoked by the user interface. This provided a true separation of the application from the user interface, one of the main advantages of the UIMS model. This allows for easier implementation of multiple interfaces and ensures that applications are truly atomic and do not fall into the trap of performing user interface generation or management functions.

Of course, this external control approach has a drawback: interface definitions must address all possible exchanges between the user and the application. This can become problematic in a large, complex application.

The model of an external control UIMS is quite similar to today's model of Web services-based Web applications where the Web service is viewed as the set of subroutines which form the application interface and the user interface is viewed as the HTML or Dynamic HTML generated by a Web application server invoking the service in constructing a page. Then it is possible to think of a Web services model of Web applications as a form of external control UIMS.

Mixed control models are an integration of internal and external control UIMS architectures. In mixed control implementations, the user interface may invoke methods in the application as subroutines while at the same time the application may directly invoke user interface methods or objects.

Most of the research in the field of UIMS strategies is focused in the 1980s and early 1990s and predates the emergence of the World Wide Web as a popular medium.

Some early commentators questioned the feasibility and desirability of the UIMS concept of separating user interface design from application design. For example, in 1987, Neches et al. [80], argued this separation strategy meant that it could limit artificial intelligence approaches to interfaces. As designers and developers want to put intelligence in the interface, it becomes desirable for the interface to have access to the specification of the application but UIMS architectures, in general, separate the specification of interfaces from the specification of applications. In addition, the interface will need knowledge of the structure as well as the internal state of the application and not all UIMS environments provide for this. They argue, ultimately, that intelligent interfaces require that UIMSs improve the connection between interfaces and applications without mitigating the benefits of UIMS approaches; the authors also acknowledge that this is no small task.

Nonetheless, even with detractors pointing out that in some domains UIMS systems may limit the ability of interface designers to achieve the designs they want, numerous notable UIMS implementations were presented to the research community in the late 1980s and early 1990s.

These implementations can share many different features:

- *A declarative language for specifying interfaces.* This can take forms based on programming languages (MIKE and Mickey [82]) to BNF grammar representations [42] and Augmented Transition Networks [52]. Other UIMS implementations have adopted petri nets (GENIUS [53]).
- *Application data model-derived user interfaces.* Many UIMS platforms step away from a pure separation of user interface design and application

design choosing an approach which allows the user interface to largely be derived from the underlying application semantics and data model.

Examples of this include MIKE and Mickey [82], Apogee [50], and User Interface Design Generator (UIDE) [38]. These UIMS platforms take different approaches to how much they automate.

- *Object-oriented approaches.* Many UIMS systems adopted an object-oriented approach including GROW [10] and GENIUS [53]. GROW, for instance, provided a hierarchical set of graphical objects with attributes where the objects could be extended by the designer, thus providing reuse and flexibility.
- *Automated user interface generation.* Numerous UIMS platforms auto-generate interfaces which are either used directly by the user or can then be refined by the designer or developer. These include Mickey [82] which generates Macintosh-like user interfaces from a Pascal-like specification, Cousin [47] which uses a value-slot mechanism to share data between the application and user interface, ITS [51] which combines style rules with a definition of dialog content to generate dialogue boxes, Chisel [103] which uses hinting to help place dialog box elements, and UIDE [38] combined with DON [56] which uses a specification and layout rules together to generate dialogues in line with a set of visual design goals stored in a knowledge base.

In the area of automatic generation of user interfaces, the approaches taken seems to cover a spectrum. At one end of the spectrum are systems where user interfaces are specified by the designer in detail (including widget selection) and generation only

involves layout of the widgets; at the other end are systems where user interfaces are defined functionally by the designer and generation involves selection of user interface widgets as well as layout of those widgets.

The experience of early UIMS systems which offered automatic generation and rendering of interfaces was that they were feasible. While no system was automating the complete design process, individual UIMS systems automated parts of the process and, according to de Baar et al. [32], writing in 1992, the evidence strongly suggested that automatic generation of user interfaces was both feasible and desirable because it relieved the designer of unnecessary work to focus on the design itself. They also noted that automatic generation of user interfaces could help to ensure consistent application of style rules.

This does not mean challenges are not present. Some particular challenges which exist, include:

- Losing control of the style of the interface. This problem was described by Olsen in his work with MIKE.
- Limiting user interfaces types. Some UIMS systems are only suited to one type of interface such as command-style interactions or modeless interfaces.

### ***3.2 Using Markup to Specify User Interfaces***

From its inception, the Web has evolved as an environment in which markup languages are used to specify user interfaces. Although HTML was never intended as a language for specifying user interfaces, but rather document structure [58], it quickly evolved into a tool for creating user interfaces and today serves that purpose on the Web

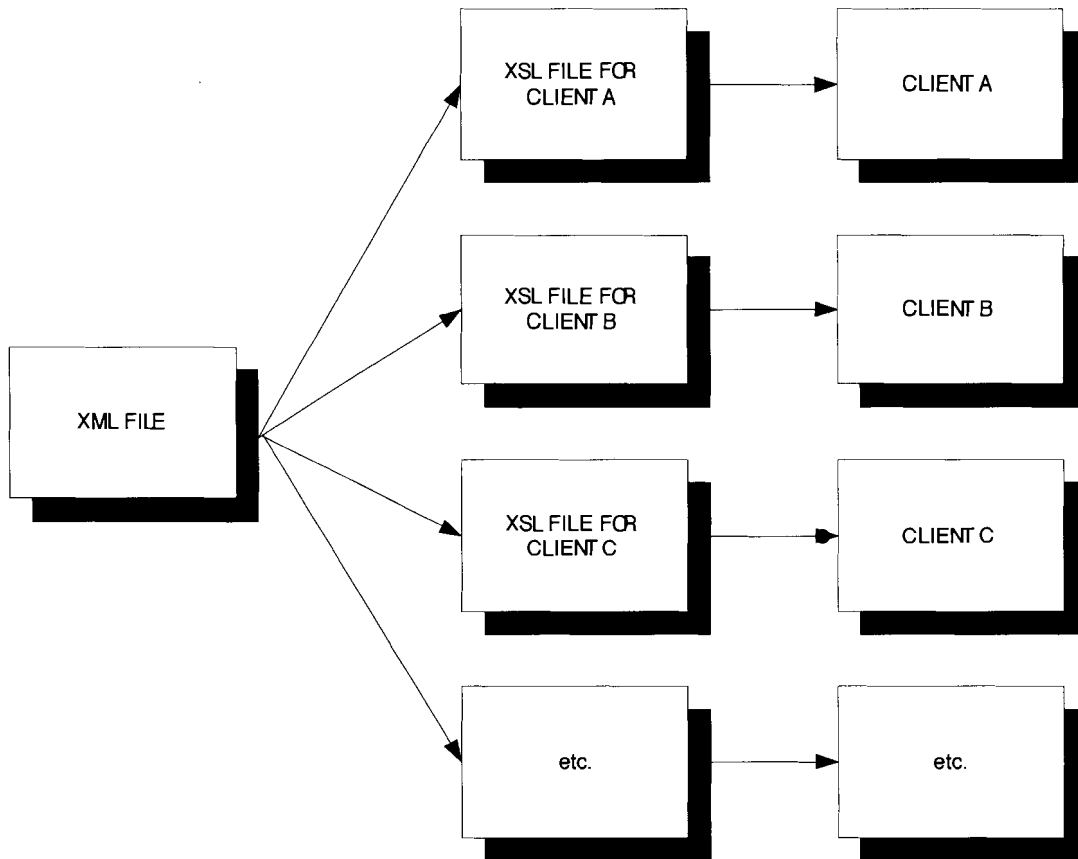
as a component of the Dynamic HTML framework used by many sites to deliver interactive user interfaces.

However, as cross-browser compatibility problems have emerged so have serious productivity and code management concerns and researchers and industry have sought alternate markup schemes to improve the development and delivery of Web applications.

### **3.2.1 Using XML to Simplify Application Delivery to Multiple Devices**

An increasing trend on the Internet which aims to address the code management problem is the use of XML-coded documents as the source for pages, documents or applications [9]. In this model, each document has a single core representation and then XSLT style transformations are written to transform the document into the various target HTML and WML formats needed for each client. In this case, the developer now has to write multiple XSLT transformation definitions instead of multiple HTML and WML documents. The XSLT language is suited to this purpose which makes the specification of the transformations easier to develop and maintain than multiple source documents. The result is an application design which looks like Figure 6.

**Figure 6** Transforming XML with XSLT for each client



While this approach provides the advantage of only having to maintain a single XML source file for each page, document or interface in an application, it still requires maintaining multiple XSL files to account for each potential client browser; this has the potential to grow unwieldy if the number of browsers is too large. If a developer must maintain an application with three distinct interfaces targeted for 10 possible clients, the following files have to be created and maintained:

- 3 XML files
- 1 XSLT file for each client: 10 files

Still, this is a more attractive solution than the previous approach without XML and XSL in terms of manageability. However, it is not the perfect solution to the problem in that it requires platform-specific XSL style sheet files.

In addition, considering the fact that the capabilities of clients to render rich interfaces varies, it may be the case that suitable interface elements for the definitions in an XML file may not be renderable in a useful way on all devices, regardless of the XSLT transformation specified for the client. Consider, for example, a control for a hierarchical tree; in a plain HTML browser there is no way to render this. In Flash, however, this control is available. There is no possible XSL transformation for a plain HTML browser to render this control even if it is called for in the source XML file.

### **3.2.2 Providing User Interface-Specific Markup with UIML**

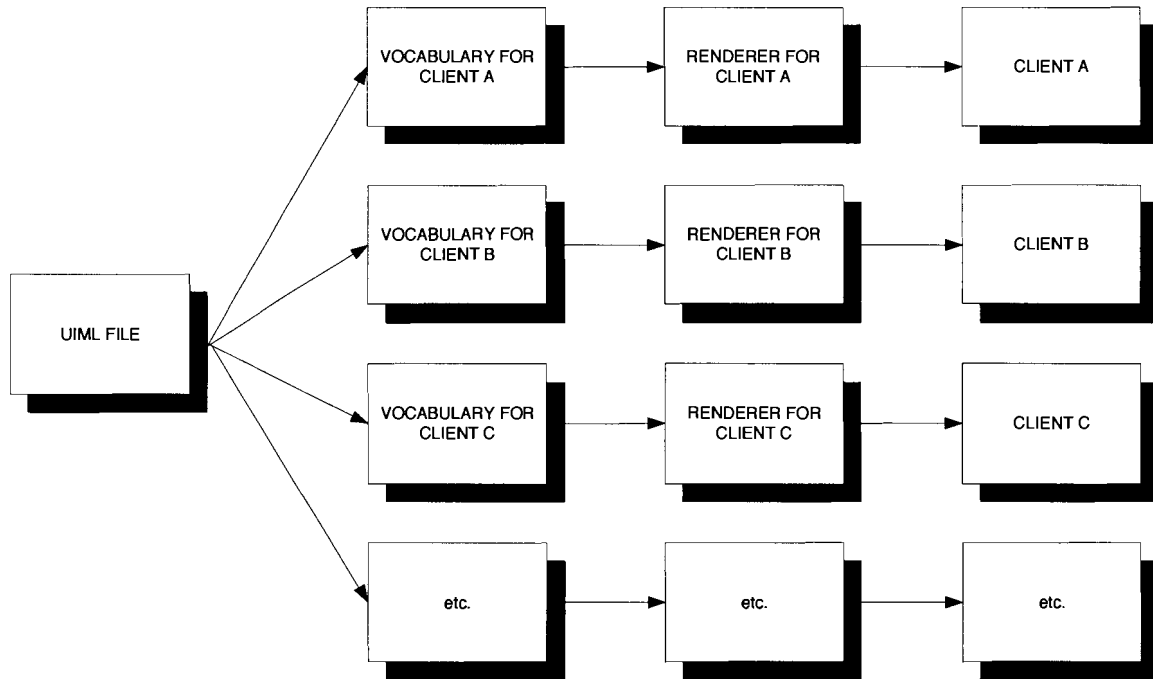
Other attempts to reduce this exponential growth in the number of files needed to develop and maintain an application using HTML/WML have also emerged. Most notably, the User Interface Markup Language (UIML) is an attempt to provide a markup language which allows for one language to be used to specify an application interface for an application deployed in multiple programming languages, on multiple devices, using multiple operating systems [1]. UIML is an XML-like meta-language which allows the creation of vocabularies for specifying user interface elements for an application. UIML serves as a generic language for defining vocabularies to allow the creation of interface definitions for any type of environment [3] [1].

In the UIML model, individual rendering engines are built to handle rendering of the UIML source code into a form suitable for a target client. Renderers exist for HTML, WML, Java and other environments.



In theory, UIML not only reduces the number of files in much the same way as XML/XSLT but also provides a markup language optimized to the user interface specification task. An application design with UIML looks like Figure 7.

**Figure 7** Using UIML to build platform-independent user interfaces



In addition, UIML and XML can be used together; in this case, document data is specified in XML and a single XSLT file is used to transform the XML into a UIML file which in turn is rendered accordingly for each device.

Still, there are limitations with current uses of UIML, including:

- The dilemma of how to handle clients of varying capability remains: it is possible to specify a user interface which must be crippled to function on some devices.
- There is a degree of platform dependence both in terms of the creation of vocabularies and in terms of the availability of rendering engines.

- It may become necessary to build new rendering engines as the diversity of target clients increases.

UIML is not a widely supported technology outside the research community which limits its utility for creating real applications for delivery to users on the Internet today.

A key aspect of UIML, its developers contend, is that it is not tied to a specific metaphor, which makes it ideally suited to the process of building multiple interfaces for an application that must be deployed in different user interface metaphors (desktop, mobile phone, voice, etc.) [1]. This differs from other attempts at creating markup languages for user interface specification which are platform and metaphor specific. The following sections discuss some of these markup languages.

### **3.2.3 The Mozilla Experience: Using XUL**

Another approach to developing markup languages for user-interface specification can be found in the development of the open-source Mozilla Web browser. Mozilla's user interface itself is specified in XML using the XML User Interface Language (XUL) created by the Mozilla development team.

XUL is designed to build portable user interfaces [33] [18] and provides mechanisms to specify most of the conventional user interface widgets and components common to modern graphical user interface applications, including:

- Input controls (text fields, check boxes, radio buttons, etc.)
- Toolbars
- Menu bars and pop-up menus

- Tabbed windows
- Trees
- Tabular data grids [33] [51] [81]

Initially, applications built using XUL, such as the Mozilla browser, embedded a rendering engine to render the XUL markup into an actual user interface. However, the project has also initiated an effort to develop the XUL Runtime Environment (XRE) [77]. XRE, once complete, should provide a single rendering engine that would allow the development of XUL-based applications without needing to build a rendering engine for each XUL-based application.

In theory, XUL provides a mechanism, when combined with the eventual delivery of XRE, for achieving an optimal situation in which an application and its interface can be delivered through the Web with the developer creating a single XUL-based user interface specification and simply delivering it to users who have the XRE engine installed.

However, it remains unclear from the current XRE documentation if:

1. The XRE engine will be platform independent or will need to be ported to different platforms and if it will be possible to port the environment to small footprint devices such as handheld computers.
2. XRE will account for differences in form factor that might make a user interface specified for the desktop unworkable on a handheld computer or other non-standard display sizes.
3. XRE combined with XUL is really only suitable to specifying interfaces for desktop-style applications or if it is suitable to a broader range of

applications including Web applications; to date, the examples of XUL-based applications, such as Mozilla, are all desktop-oriented.

Functionally, XUL is designed specifically to specify user interfaces; it does not provide mechanisms to define how to link interface elements and actions to behind-the-scenes operations and services. For instance, to make a Web application function, it is necessary to be able to specify how each interface element and user interaction with the interface is linked to HTTP GET or POST operations or to Web service invocations.

Nonetheless, XUL is a clear attempt to define a user interface-specific markup language (as opposed to UIML which is a meta-language for defining other languages) which has been shown to be usable for specifying even complex user interfaces (as in the case of the Mozilla browser itself).

#### **3.2.4 XForms: A Proposed Standard for Web Forms**

XForms is a recommendation of the World Wide Web Consortium for the next generation of Web-based forms [124]. At the core of XForms is the notion of separating descriptions of what a form does from the way it should appear. This is intended to make it easier to deploy forms to a range of clients with different capabilities and limitations.

In principle, a form defined using the XML-based XForms model could then be rendered in HTML for standard Web browsers, in WML for mobile phones, in the XForms User Interface (a standard set of controls designed to replace today's HTML-based form controls), and any other client platform.

The XForms standard offers several interesting features:

- A focus on form definition around data collection.

- Support for workflow auto-fill and pre-fill of forms.
- A submission protocol to handle communication with a back-end which supports suspend and resuming form completion.

At the present time, a public test suite and several implementations of XForms are available. However, major browsers such as Internet Explorer have no native support for XForms requiring users to install plug-ins in their browser or developers to transform XForms into HTML at the server.

Although it is unclear to what extent XForms is being used on the server to generate HTML forms which are seen in today's Web site, an informal survey of major Web sites doesn't reveal any sites which presently use XForms at the client to provide a richer form experience to users.

In addition, XForms does not explicitly address the question of rendering: how to effectively render a given form on a given display in a way which is functional and usable.

### **3.2.5 Other Markup Languages**

Other markup languages for user interface specification exist:

- WML (Wireless Markup Language) [92] [2]: WML is a language designed specifically for the creation of user interfaces for mobile devices such as mobile phones.
- HDML (Handheld Device Markup Language) [2]: HDML is a language for the creation of user interfaces for mobile devices. It is a proprietary language implemented specifically on browsers from OpenWave.

- CHTML (Compact HTML) [54]: Compact HTML is a stripped-down version of HTML targeting resource-limited devices. It lacks support for JPEG images, tables, background colours and images, style sheets and much more.
- VoiceXML (Voice Extensible Markup Language) [115]: VoiceXML is an XML-based markup language for developing speech-enabled applications for delivery on the Internet.

### ***3.3 Addressing the Problem of Small Screens***

Aside from the question of what languages are suited to specifying user interface definitions and optimizing the process of specifying user interfaces for deployment to multiple platforms, a second fundamental question emerges in the delivery of Web applications in a modern environment consisting of varied devices with both full-size and small displays: how to optimize the display of content and other user interface elements on different size displays.

To date, most Web content has been developed with desktop computer users in mind and, accordingly, much of the literature surrounding this question focuses on techniques for the automatic transformation of content for display on small devices as well as techniques to make it easy for users to view and navigate large data sets on small displays.

#### **3.3.1 Automatic Transformation of Content**

In the area of mobile Web access, a broad body of research has emerged since the mid 1990s addressing the question of how to render existing Web content in a manner suitable for easy access on devices with small screens. Typically, this research looks at

mechanisms for automatically transforming existing Web content as it is requested by the client and is not concerned with the visualization techniques discussed in the previous section.

Trevor et al. [112], identify four main techniques for presenting Web content on small devices:

1. **Scaling:** Existing pages are simply scaled to fit as best possible on smaller screens. This technique is used by Pocket Internet Explorer on the Pocket PC operating system from Microsoft. The largest drawback of this approach identified by Trevor et al., is that it sacrifices readability.
2. **Manual Authoring:** This approach, outlined earlier in this document, is to simply produce different implementations of an application interface for each target client system. This produces good results but is an inefficient use of human resources for development.
3. **Transducing:** This approach involves the automatic translation of content into a form appropriate for the target device. For instance, there are software packages available which attempt to convert HTML to WML in real-time as pages are requested [93] [20].
4. **Transforming:** This approach involves modifying content to transform the structure, user experience and navigation. Often this will include a transducing step as well.

The process of transforming content has received attention in numerous studies. Four approaches to this transformation are outlined below.

### ***3.3.1.1 Power Browser***

Power Browser, a project by Buyukkokten et al. [23], is an attempt to address the delivery of Web content to PDAs by using a transformation model which displays hierarchical link summaries of Web content. In attempting this, the developers were trying to address a problem they noted in small screen devices: users are less willing to follow links with small devices—in other words, less spontaneous browsing occurs.

Power Browser is a proxy solution which performs Web requests on behalf of client handheld devices. It downloads the requested document and analyzes the link structure of the document, preprocesses the document and selects the data to display as a hierarchical summary of links, including links found on linked pages.

In its preprocessing, Power Browser discards white space, ignores text attributes and folds tables into text blocks and provides the user the ability to order links following the original document, in alphabetical order or in a page ranked format.

When a user navigates to a page they are interested in, Power Browser uses a summarization facility to partition the page into semantic textual units and then, within each textual unit, attempts to glean a hierarchical structure. Then users can expand and contract these various units to make a large page navigable on a smaller screen [24].

Especially notable is Power Browser's form summary technique designed to make Web forms usable on small screens. Power Browser extracts the textual labels that prompt users for input without displaying the form widgets. These labels are displayed in a list and the user can select each label individually to display the prompt and input information for that widget. Their algorithm reports a 95% success rate in selecting the appropriate labels for the form elements from the source HTML.



Although Buyukkokten et al. do not report on user studies specifically evaluating their form summarization technique, visual inspection of sample results shows that the technique effectively reduce large forms into a format which can be displayed on small PDA screens; at the same time, though, it is a technique specifically targeted at small screens and is not aimed at providing general-purpose transformations for a wide range of client screen dimensions.

### ***3.3.1.2 Digestor***

Digestor [15] takes a Web page and splits it into multiple Web pages suited for smaller displays and adds new navigation links. The developers found that this works well down to a threshold: as screens get too small, the user interface breaks down and becomes hard for users to understand, manage and navigate.

### ***3.3.1.3 M-Links***

The M-Links project attempts to present Web sites on small screens by transforming both the document structure and altering the traditional navigational model of Web browsing [112] [100]. M-Links is a proxy server architecture which handles Web requests on behalf of client handhelds and transforms the incoming content before sending it back to the browser. These transformations take into account the markup language supported by the client so that the resulting interface is returned in the appropriate language chosen from HDML, WML, CHTML and a subset of HTML.

The concept behind M-Links is to separate link structure from content and, in doing so, delineate the process of navigating and the process of acting on content. Their technique involves extracting links from a page and presenting them as a list to the user who can select any link in the list to navigate to that page. Once the user reaches the

desired page they can switch to a mode where they can act on the content of that page.

Actions include, but are not limited to:

- Viewing the content
- Printing the page
- Faxing the page
- Bookmarking the page

This differs from traditional browsing where a user is viewing navigational links mixed in with page content [100]. Under this model, the navigational process breaks down to three steps:

1. User requests a link to visit
2. User is presented a list of links and drills down by repeating step 1
3. User chooses destination content and is presented a list of possible actions

In this process, links are not simply presented as a list in the order they appear in the original document. Instead links are categorized automatically and grouped in logical ways based on the structure and content of the original document. In addition, links are extracted which might not be actual hyperlinks in the document; for instance, phone numbers and addresses for contact information are automatically detected and turned into links.

M-Links manages to allow users to use this navigational paradigm with only four buttons on a handheld device. On larger displays, M-Links provides no options for a

more standard navigational paradigm which forces all users on all clients to use the same navigational model.

#### **3.3.1.4 WEST**

Bjork et al., [16] propose a model for navigating Web content on PDA devices which combines text reduction and link extraction techniques, similar to Power Browser, Digestor and M-Links, and add a focus in context style of visual presentation technique. The resulting browser was termed WEST (Web Browser for Small Terminals).

The basic algorithm used was implemented on a proxy server. First, an HTML page was preprocessed in three steps:

1. The page was divided into smaller pages or cards; this is similar to the Digestor approach of splitting a page into multiple smaller pages. These cards were then grouped into decks.
2. The content of each card was summarized as a set of keywords.
3. All hyperlinks on each card were extracted.

Users then had the choice of three possible focus-in-context views:

1. Thumbnails of the top cards of all decks were displayed in a grid format.
2. Keywords summaries were presented on each card.
3. Extracted links were presented on each card.

In each view the user could zoom in on a single deck to view fully readable content while the remaining decks would surround the zoomed deck. This tile-based visualization and navigation technique is known as flip zooming.

Decks were used to limit the number of cards on the screen at one time to seven to allow the context objects to be displayed at a reasonable size and to allow ordering to be retained so that the fourth card in the sequence would always be fourth whether it sat in the focus position or in a context position.

To test the usability of the technique, Bjork et al., conducted a user study to compare this technique to a traditional style browser running on a Palm PDA handheld. Results indicate that users rated WEST higher than the traditional style browser and that searching through content was easier. At the same time, the flip zooming technique was not immediately intuitive to users who had some initial learning curve difficulties.

### **3.3.2 Displaying and Navigating Large Data Sets on Small Displays**

A significant dilemma of small screens is how to present and provide for large content sets. This is not a problem unique to small screens, in fact. The problem also exists when trying to view extremely large data sets on standard desktop displays as well.

Numerous algorithms exist for attempting to ease the problem of display and navigation of large data sets. Two main classes of techniques exist:

1. **Fish Eyes and Distortion Techniques:** The concept behind fish eye lenses visualization and navigation is that the greatest level of detail should be provided for the content of interest, content close to it should have somewhat less detail and so on; as content is further away from the central content of interest, less detail should be presented. A generalized description of this strategy was presented in 1986 by Furnas [39]. Several subsequent studies have built on Furnas' work with fish eye views. Sarkar et al. [99], for instance, envisage a rubber sheet metaphor while Roberston

and Mackinlay [95] proposed an alternative to the fish eye model specifically addressed at document visualization.

2. **Zooming and Overview Techniques:** In 1995, Kaptelinin [55] compared four navigation and visualization techniques for navigating large data sets; these were scroll bars, direct dragging of the data set, alternately switching between a map and a close-up view, and an extension of the map to provide context in the map identifying the current close-up view. Baudisch et al. [11], also compared several navigation and visualization techniques for large data sets including alternating between zoomed and high-level overview views, overview views with a second window providing detail of a region of the overview, and focus plus context views using specialized hardware.

These two classes parallel two of the three category's in Spence's taxonomy of graphical presentation [104].

Although not specific to small screen displays, these techniques address the problem of extremely large content sets which are hard to render and navigate on any size device.

## CHAPTER FOUR: SCALABLE USER INTERFACES

Our Scalable User Interfaces (SUI) has been implemented using the latest versions of Flash and Flash Player. The current version of the Flash Player is Flash Player 7; development for Flash Player 7 is done using the new Flash MX 2004 and Flash MX Professional 2004 development tools. As with previous versions of the Flash Player, Flash Player 7 is backwards compatible and can play Flash movies created with previous versions of the Flash development tools.

### ***4.1 Flash***

As mentioned earlier, the Flash environment is divided into two main components:

1. *Flash Player*: Flash player is a runtime engine which can process and run compiled Flash movies. The term *movie* simply refers to a compiled Flash file which can be run by the Flash Player and is a holdover to the early days of Flash when Flash was primarily used as an animation tool for the Web. The Flash Player is implemented by Macromedia as a native binary executable program on numerous different platforms. The current version is version 7 and is available as a stand-alone player on major computing platforms such as Windows, Mac OS X and Linux as well as a plug-in for Internet Explorer and Mozilla-based browsers as well as in the form of embedded players for hand-held devices and mobile phones. A compelling feature of the Flash Player is the feature consistency across platforms of

the player; a given Flash movie should run identically on all platforms with the same version of the Flash player.

2. *Flash*: Typically, Flash movies are developed with Flash which is an integrated development environment providing coding and design-level tools for building Flash movies, testing them and, finally compiling them into files which can execute in the Flash Player. The current version of Flash resembles a traditional integrated development environment such as Visual Basic with added graphics and animation tools.

Early development of SUI took place using Flash MX and Flash Player 6. In 2003, in the midst of the work on this project, Macromedia released Flash MX 2004 and Flash Player 7 which offers improved data and widget support. Development of SUI moved to Flash MX 2004 and Flash Player 7 at that time. (In this thesis, Flash MX 2004, Flash MX and Flash will be used interchangeably).

The Flash MX 2004 development environment and the Flash Player 7 collectively offer several features which are essential to making SUI viable to implement in Flash:

- A component architecture
- Built-in XML Parsing
- Cross-Platform Compatibility

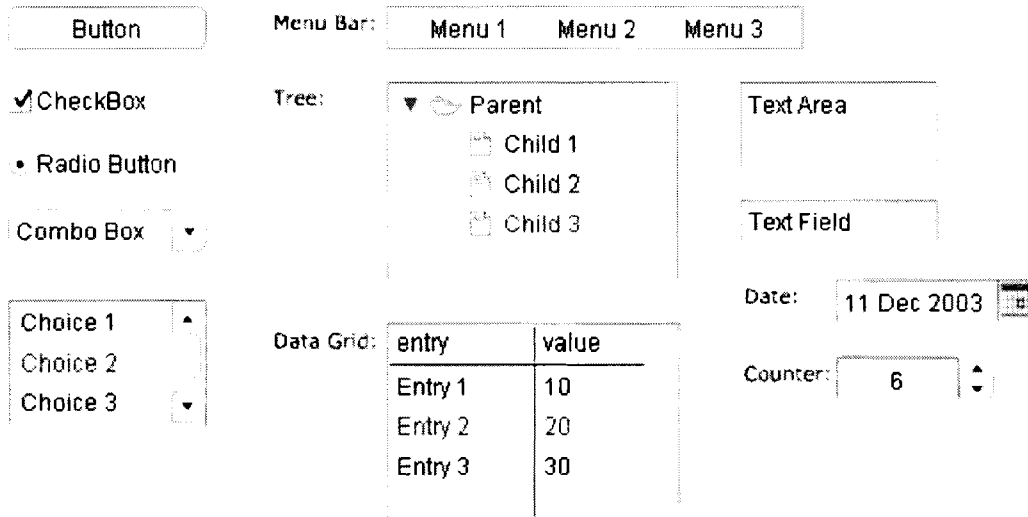
#### **4.1.1. Component Architecture**

Flash MX and Flash Player 7 provide a component architecture. This allows developers to create and reuse user interface or application logic components which can

then simply be placed in applications being developed in Flash MX. These components can be extended and customized by other developers

With the introduction of the component architecture in Flash MX, Flash now offers a set of pre-existing user interface components which can be integrated into Flash movies. These components include standard interface components such as text fields, buttons and selection lists as well as more complex components such as tree controls, data grids and tabbed panels. Figure 8 provides examples of these components.

**Figure 8** Flash MX 2004's User Interface Components



SUI relies on these components to provide the core functionality for its widget implementations. The SUI widgets are generally built as a set of functions which manipulate one of these pre-existing Flash MX user interface components.

#### 4.1.2 Built-In XML Parsing

With Flash MX, Macromedia introduced XML support into the Flash development environment. Flash MX provides an XML object which can be used to parse and manipulate XML. Using the object, it is possible to load an XML document



and traverse the tree of nodes in the XML document without every having to directly parse the XML source code.

As SUI will rely on an XML-based markup language, the XML object allows the SUI rendering engine to easily access and manipulate documents in XML. When XML is loaded from a file, or XML source code is provided in-line in ActionScript, a node tree is created with an XML object for each node in the tree. It is possible using properties and methods of the XML object to traverse and manipulate the tree.

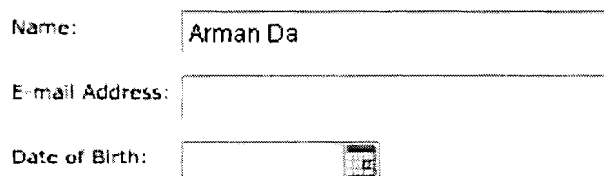
### 4.1.3 Cross-Platform Compatibility

As mentioned previously, one of Flash's most compelling features in the area of Web applications tools is its cross-platform compatibility. The Flash Player is available on a wide range of devices, including:

- Desktop computer operating systems (Windows, Mac OS, Unix, Linux)
- Handheld PCs
- Mobile phones

What is particularly noteworthy about this cross-platform compatibility is the consistency of appearance of Flash movie files playing on the Flash Player on these various platforms. Figures 9 and 10 show the same file playing on the Flash Player on a handheld device and Linux. In both cases, the appearance of the interface elements is consistent.

**Figure 9**      **A simple Flash movie containing a form on a handheld device**

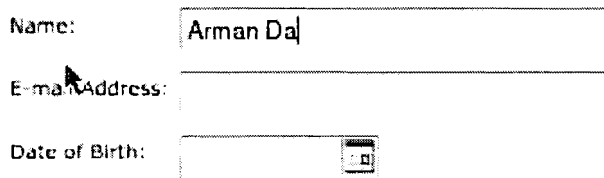


Name:

E-mail Address:

Date of Birth:

**Figure 10**      **A simple Flash movie containing a form on Linux**



Name: Arman Da

E-mail Address:

Date of Birth:

#### **4.1.4 Flash on the Server: Macromedia Flex**

Late in the development of this thesis project, Macromedia publicly announced a new Flash-related technology named Flex and released the software into limited public beta testing.

Flex is an initiative to provide for server-based development of Flash movies for enterprise data applications. Since the release of Flash MX, Macromedia has clearly been attempting to position Flash as a platform for building interactive interfaces to back-end data applications deployed on the Web. However, the development environment of Flash MX is inconsistent with the normal development cycles and processes used in the development of Web-based enterprise data applications.

In typical Web development processes, the specification of the user interface is done with HTML and JavaScript and this code is tightly integrated with application logic code written in ColdFusion, PHP, ASP.NET or similar languages.

However, Flash requires a development paradigm more similar to desktop application development. With Flash, the client interface of an application is designed visually and the code necessary to support that interface is added to the client with an integrated development environment tool. The server-side application logic is written and deployed separately from this client.

This moves the development of the client interface into the realm of graphical development typically accomplished in a drag-and-drop environment; no markup language is used. Once the interface is developed, the application logic is then implemented in ActionScript and the movies are compiled and deployed to the server for delivery to the user. Developers must, therefore, possess expertise in the use of the Flash MX development tool.

Macromedia Flex provides a completely new approach to deploying the application logic and presentation layers for Flash-based Web data applications. Macromedia Flex is a server-side application server which allows the creation of Flash user interfaces using a new markup language, MXML (MX Markup Language), and CSS while simple ActionScript is used to handle application logic. These MXML files are deployed as plain-text files on the server and are compiled by Flex at request time. This makes the development process similar to using HTML, CSS and JavaScript and provides a development model which is familiar to Web application developers.

This means developers continue to use the same development models they used with HTML but when the individual files are requested by the user they are compiled into Flash movies on the fly and delivered to the Flash Player on the client.

For the purposes of SUI and this thesis, Flex is not well suited. The goal of SUI is to provide a single client application which can render interactive forms specified in a single markup file without dependence on the server to execute any of the rendering logic. The key is that the exact same client application runs on any client platform and uses exactly the same markup to render the form in a manner most suited to the display real estate available in the client.

With Flex, the layout of the form would be specified in the MXML file and there would be no straightforward way to account for the size of the client display and adjust the rendering strategy accordingly.

With Flex, the logic approach would be to shift the rendering algorithms to the server and build a custom MXML file from a SUI markup file and deliver a custom Flash to the client for each request. This would allow the rendering algorithms to run on the server where, typically, much more processing power is available and more sophisticated rendering algorithms could, therefore, be created.

However, this runs counter to the write once, run anywhere principle. Under SUI, there are no server platform dependencies. A SUI markup file can be served by any HTTP server as well as being dynamically generated on any Web application server. With Flex, the SUI markup files could only be served by Flex-based servers.

Nonetheless, Flex is a critical new component of the Flash environment and may in the future offer insights into the evolution of the markup language used in SUI.

#### ***4.2 SUIML: Markup for SUI***

A markup language for SUI has been created called the SUI Markup Language (SUIML). SUIML is loosely based on the XUL markup language for specifying application interfaces in the Mozilla project. Specifically, the set of tags used includes a subset of tags from XUL plus additional SUI-specific tags. However, at the level of tag attributes and rules regarding tag embedding, SUI is not based on XUL.

SUIML follows the basic syntax requirements of XML-based languages. Specifically, all tags must be closed using either of the two standard forms:

```
<tag attributes ...> ... </tag>
```

or

```
<tag attributes ... />
```

The following section provides a high-level overview of SUIML and the full language is specified in Appendix 1.

#### 4.2.1 The Tag Set

The tag set for SUIML can be divided into three main groups:

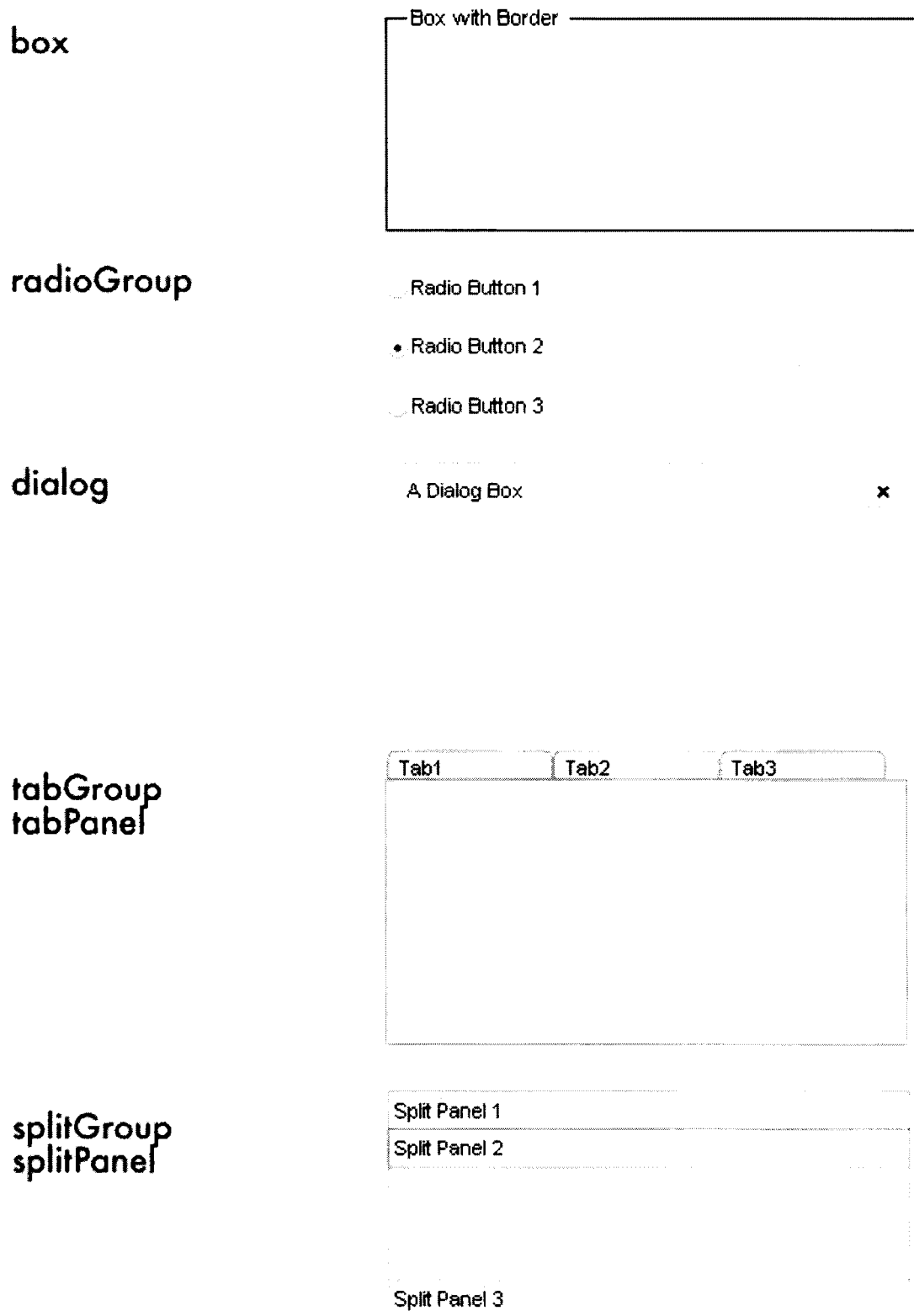
1. Tags which define containers which hold one or more widget tags.

Examples of these containers are illustrated in Figure 11.

- **box**: a generic rectangular box in which a group of widgets can be placed so that the widgets are treated as a single entity for the purposes of form layout.
- **radioGroup**: a group of related radio buttons; only one button in a group of radio buttons can be selected at any given time.
- **dialog**: a dialog box is a group of associated widgets which can be displayed in a layer above the main form in much the same way as a floating window.
- **tabGroup**: a group of related tab panels; only one tab panel can be visible, at the top of the stack of tab panels, at any given time.
- **tabPanel**: a tab panel contains the set of widgets displayed on any given page in a tab group.

- `splitGroup`: a group of two related split panels; a slider can be moved between the panels to control how much of each panel is visible at any given time.
- `splitPanel`: a split panel contains the set of widgets displayed in a specific panel in a split group.

Figure 11 Container-level elements in SUI



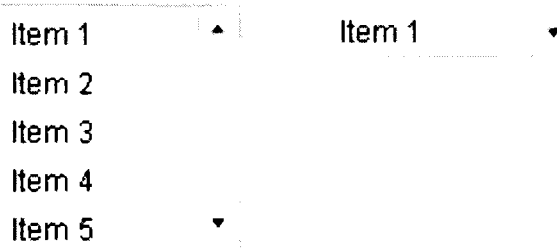


2. Tags which define individual widgets. Examples of these widgets are illustrated in Figure 12.

- `button`: a regular form button
- `checkbox`: a check box
- `image`: an image
- `select`: a selection list such as the selection list boxes or drop-down selection lists found in HTML
- `item`: an entry in a selection list
- `radio`: a radio button in a radio group
- `label`: a text label which cannot be edited or altered by the user
- `text`: a text field



Figure 12 Widget-level elements in SUI

<b>button</b>	
<b>checkbox</b>	<input checked="" type="checkbox"/> CheckBox
<b>image</b>	
<b>select item</b>	
<b>radio</b>	<input type="radio"/> Radio Button
<b>label</b>	Label
<b>text</b>	<input type="text"/> Text field

3. Tags which define connectivity to back-end Web services.

- **application**: used to specify the back-end Web service a form will interact with.

It is important to note that recursion is possible in that containers can be embedded as widgets within other containers creating, in theory, a nesting of containers to any depth.

### 4.2.2 Required Attributes

Every tag requires, at a minimum, the following two attributes:

1. **name**: every widget must be named. This allows for referencing of specific widgets which is important for handling data returned from invoking Web service methods.
2. **label**: every widget should have a label assigned to it. The label is displayed to describe the form element. This is handled in different ways. For instance, for a text field, the label serves as a prompt for the text field while with a tab panel the label appears on the tab associated with the tab panel in the tab group.

### 4.2.3 Tag-specific Attributes

Many tags can take additional attributes other than the two required attributes.

These attributes can affect appearance or provide event handling to respond to user actions:

- **border**: some widgets can have an optional border.
- **clickMethod**: when a widget can be clicked by the user this attribute specifies a Web service method to invoke upon a click by the user.
- **value**: when a widget can have a default initial value, this attribute specifies the value.
- **selectMethod**: when a widget can be selected by the user this attribute specifies a Web service method to invoke upon selection by the user.

- `unselectMethod`: when a widget can be selected by the user, this attribute specifies a Web service method to invoke upon deselection by the user.
- `openMethod`: when a widget can be opened by the user, this attribute specifies a Web service method to invoke upon opening by the user.
- `closeMethod`: when a widget can be closed by the user, this attribute specifies a Web service method to invoke when the user closes the widget.
- `scale`: some widgets, such as images, can be scaled; this attributes specifies the scaling as a percentage of the default widget size.
- `size`: specifies the number of characters which can be displayed in a line in a text field.
- `lines`: specifies the number of lines to display in a text field.
- `changeMethod`: when the value of a widget can be changed by the user, this attribute specifies a Web service method to invoke upon such a change by the user.

It is important to note that optional attributes which control appearance do not represent absolutes. That is, when sufficient display space is available to render widgets in their normal form, then these display attributes apply. However, if alternative rendering methods need to be used, the display attributes may be ignored. For instance, in a minimal layout form for a small screen display, the `size` and `lines` attributes for a text field may be ignored to make display of the text field possible.

#### 4.2.4 Managing SOAP Services Callbacks

For any Web form to be useful, it must be possible to both pass data provided by the user in the form to a server for processing as well as respond to data returned by the server to the form for display for the user. This allows a Web form to provide an interactive experience for the user. For instance, consider a simple example of a form with three widgets:

1. A text field where the user can enter a stock ticker symbol.
2. A button a user can click to submit the stock ticker symbol.
3. A text field used to display the current value of the stock the user specified with a ticker symbol.

Use of the form involves the following workflow:

1. The user enters a stock ticker symbol in the first field.
2. The user clicks on the button.
3. The value from the first text field is sent to the server.
4. The server returns the stock's value to the form.
5. The form displays the stock value in the second field.

The key here is steps 3 and 4 where the form must connect to the server and communicate. Without these steps, the form serves no real purpose because it cannot connect the client to the server.

The SUI model leverages the emergence of Web services standards and particularly those based on the SOAP protocol to link the SUI forms to back-end services. Using Flash, the SUI rendering engine can invoke SOAP-based, XML Web

services running any standard Web application server platform including Java, .NET and ColdFusion servers. These Web services expose methods which the SUI rendering can invoke and these methods can be built to return data to the Flash-based rendering engine.

For the purposes of the SUI environment, SOAP will be used in a simple two-way interaction with these Web service methods. When an event occurs in the form which triggers a Web service method, an associative array is built containing the widget names for all widgets in the form as well as any values available for the widgets; it is necessary to pass all widgets to the Web service as a Web service may require all data in the form to perform its work. This is passed, along with the name of the form field where the event occurred, to the Web service method as arguments.

The Web service method, after finishing processing, must then return another associative array. This array should contain one entry for each form field which must be changed or updated as a result of the processing. The SUI engine can then update the form based on this associative array.

It is important to note that Web services technology does not specify the type of data which must be passed to a Web service nor the type of data structures to use; instead SOAP allows the specification of what data is being passed in a way that can be used by the Web service. Each Web service is developed with its own application-specific requirements regarding the nature of type of data which must be provided to the service. In the SUI case, the model is not intended to be general-purposes; an SUI form would not consume any arbitrary Web service but rather Web services would be built specifically to communicate with SUI forms.

To illustrate this, consider the stock ticker with form fields named:

1. First text field: `stockSymbol`
2. Button: `stockButton`
3. Second text field: `stockValue`

If the user enters MACR as the stock symbol in the first text field and then submits the field, an associative array is built containing the following entries:

- `stockSymbol`: MACR
- `stockButton`: no value
- `stockValue`: no value

This associative array along with the name of the button, which is where the user triggers the click event, are passed as arguments to the Web service method. After processing, the method returns an associative array with a single entry. Assuming the stock's value is \$10.50 then that array would contain the following entry:

- `stockValue`: 10.50

After receiving the array, the SUI rendering engine will update the `stockValue` text field to display 10.50. Figure 13 shows the before and after appearance of the form rendered with SUI in its full, regular rendering used when sufficient space is available.

**Figure 13** The stock application before and after the user enters data and clicks the button

Stock Symbol	Stock Symbol
<input type="text"/>	<input type="text" value="MACR"/>
Get Value	Get Value
Stock Value	Stock Value
<input type="text"/>	<input type="text" value="10.50"/>

#### **4.2.5 Current Implementation**

The purpose of this thesis is to provide for experimentation with the rendering algorithms. Accordingly, the prototype implementation includes a subset of the widgets; in addition, the callback mechanism is not implemented as it has no direct bearing on rendering of the forms themselves. The implemented tags include individual widgets and containers:

- `box`
- `button`
- `checkbox`
- `image`
- `item`
- `label`
- `radio`
- `radioGroup`
- `text`
- `select`

#### ***4.3 Transformation Techniques for SUI***

The domain of this project is limited to Web data applications. These applications consist of two main components: forms for collecting data from users and forms, tables and lists for presenting results to users.

Therefore, the transformation technique adopted for the rendering model component built for SUI must address several key questions:

1. How to render forms as the screen gets smaller?
2. How to display large result sets to the user?

#### **4.3.1 Rendering Forms**

SUI needs to account for three possible conditions in all rendering:

1. Cases where the available display can comfortably display the required content.
2. Cases where the required content does not comfortably fit in the display but where the display still offers space for sophisticated presentation. For instance, it might be possible to split a document and offer navigation tools as well without overwhelming the available display space.
3. Cases where the screen is so small, many techniques break down. For instance, in the case of Digestor [15], it was found that the technique of deconstructing a page to multiple pages and adding navigation controls became unusable when the screen became smaller than a threshold.

For forms, a three-tiered rendering and transformation strategy appears to offer promise:

1. If a form fits comfortably in the display, render the entire form. In the current SUI implementation, this is referred to as “regular” rendering.
2. If the display allows it, deconstruct the form into component parts and display them with a focus-in-context visualization and navigation model



similar to Bjork et al.'s work with WEST [16]. This visualization and navigation model provides an effective balance between providing usable form widgets and generating forms which are too large for the screen in the case of moderately small displays. In the current SUI implementation, this is referred to as “card” rendering.

3. If the screen is too small for a WEST-style focus-in-context card display, implement Buyukkokten et al.'s form summarization technique and render the form as a list of text labels which can individually be expanded into widgets [24]. In the current SUI implementation, this is referred to as “minimal” rendering.

The decision of which technique to use is derived from a combination of the actual limitations of screen space on the client and the actual form definition itself. After all, a single-widget form may display in its entirety on the smallest display but a switch to two entry widgets will mean the smaller displays will have to shift to one of the two transformation techniques and as each additional widget is added to the form, the threshold in terms of screen size at which the two transformation techniques come into place will be raised. Examples of the three rendering styles are illustrated in Figure 14.

**Figure 14** The same form using the regular, card and minimal layout styles (from left to right)

The figure illustrates three different rendering styles for a form with the following fields: First Name (Arman), Last Name (Danesh), E-mail Address (me@my.domain), and Country.

- Regular style (left):** Each field is represented by a separate text input widget with its label above it.
- Card style (middle):** The form is contained within a single rectangular frame. The labels 'First Name' and 'Last Name' are positioned above their respective input fields. The 'E-mail Address' and 'Country' labels are positioned to the left of their respective input fields.
- Minimal style (right):** The form is rendered as a vertical list of text labels. Each label is followed by a small, square expandable widget (indicated by a triangle) that can be clicked to reveal the corresponding input field.

### 4.3.2 Displaying Result Sets

Result sets typically are displayed in two different ways: as tables or as lists. Here result sets refer to tabular results distinct from specific data values being returned to be displayed in an individual widget in a form. For instance, if a search form is submitted to query a database for all records matching a given set of criteria, the data returned will be a result set containing rows and columns reflecting the relevant data from the database. This data would not be rendered in the original search form but would be displayed in tabular or list form.

In this type of case, several conditions might exist:

1. If a list or table fits comfortably on the display, just display it without any transformation.
2. If a table or list fits the display horizontally but requires vertical scrolling, present a scrolling presentation. Thresholds will need to be established to limit the number of screens of depth allowed. After all, if a page is three screens deep it is fairly easy to retain a sense of place and context when scrolling. By contrast, if a page is 100 screens deep it should get harder to keep a sense of location and context just as the case with zooming techniques where a user zoomed into a portion of a data set loses the context of the entire data set.
3. If a table or list requires horizontal and vertical scrolling or excessive vertical scrolling alone, implement a focus-in-context browsing strategy based on WEST [16]: the list can be broken down into groups based on

one of the key data fields and then a focus-in-context view can be presented.

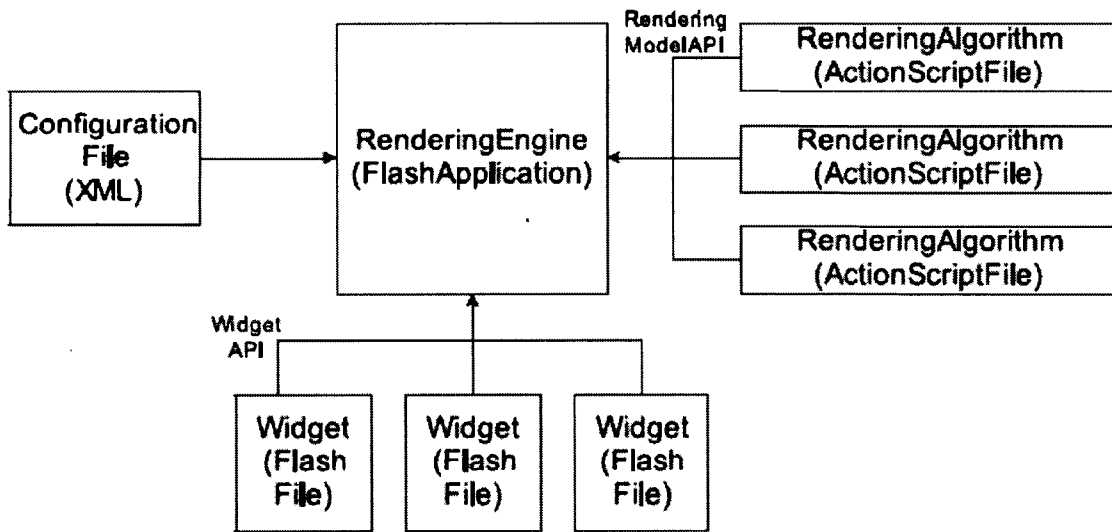
4. If the display doesn't allow a WEST-style focus-in-context presentation, implement a summarization technique similar to that user in Power Browser [24]. Each entry is summarized with identifying data (such as the first word or two in a text field); these summarized entries are listed sequentially on the display and users can selectively expand individual entries to view their full contents and edit them. If these summarized entries still create too many screens of depth, block-level widgets such as tab panels or boxes can be used to group entries into a two-level hierarchical tree in order to reduce screen depth.

#### ***4.4 The SUI Rendering Engine API and Object Model***

The SUI model is designed to provide a modular framework in which new, alternative rendering models can be implemented; this will allow future experimentation with new rendering. In addition, widgets are implemented in a modular fashion which makes it easy to add new widgets or change the implementation of the widgets.

The result is an architecture which looks like Figure 15.

Figure 15 SUP's Modular Architecture



In the figure, each rendering algorithm is implemented as an ActionScript library in an external script file which is included in the main Flash movie at the time the rendering engine movie is compiled. By contrast, the widgets are implemented as individually compiled Flash movies which can be included into a rendered form at runtime.

The main rendering engine requires that each rendering algorithm or model, as well as each as each widget movie, expose a set of standard methods which the rendering engine can use to access these components.

#### 4.4.1 The Main Rendering Engine

The main rendering engine movie fulfils several main tasks; in doing this, it relies on a configuration file discussed in Section 4.4.5 which specifies available rendering algorithms, scaling limits and other configuration information needed to execute the rendering. Rendering proceeds as follows:

1. It must determine the dimensions of the display that is being used to render the form.
2. It must open the appropriate SUIML file containing the specification of the form to be rendered.
3. It must parse and traverse the node hierarchy and create containers for each widget in the form; the appropriate widget files should be loaded into the containers but the containers should be left hidden and not visible pending future manipulation of the containers prior to final rendering and display to the user.
4. It must then invoke the rendering algorithms to determine which to use to render the form. This step works as follows for each algorithm in turn:
  - a. Request the algorithm to render the layout of the form and return the width and height. If the width and height fit in the available space, stop here and move on to Step 5
  - b. If the form does not fit, progressively reduce the scaling of the form up to a configurable scaling threshold to see if it is possible to fit the form in the space available using the current layout algorithm. If an acceptable scaling is found, stop here and move on to Step 5
  - c. If the layout method does not work, move on to the next available algorithm and return to Step A
  - d. If no more algorithms are available, apply the maximum scaling reduction allowed to the last rendering algorithm used and move

on to Step 5. The maximum scaling is configurable in the configuration file.

5. It must make the widgets visible for the user.
6. Finally, the rendering engine must respond to any events for which event handlers have been specified in the SUIML code and invoke the necessary Web service methods.

#### **4.4.2 Rendering Model API**

Each rendering algorithm is implemented as an independent ActionScript file. An ActionScript file is simply a text file containing ActionScript code which can be included in a Flash movie at compile time. In this way, the ActionScript code is compiled as part of the movie.

The rendering algorithm API files provide a single required method: `layout`. The rendering engine passes this method an array; each entry in the array is an object containing information about a widget to be included in the layout.

The `layout` method must decide how to place the widgets on the page and set the appropriate rendering style for each widget and then return an object with two properties: the final height and width of the resulting layout.

The rendering algorithm's ActionScript file can contain other methods which it uses internally and can even include and reference other ActionScript files if needed for the modularity and manageability of the code; however, only the `layout` method will be directly invoked by the main rendering engine movie.

A high-level outline of a typical rendering algorithm's layout method would be:

1. Receive a list of widgets as an array.
2. Loop through array and determine placement of the nodes.
3. For each node, obtain a reference to the movie clip for the node and place the movie clip. A movie clip is an object in a Flash movie which can contain another Flash movie, including an external Flash movie. As discussed in the next section, each widget is implemented as its own, separate Flash movie and these will be loaded into movie clips for rendering the nodes.
4. Determine the total width and height for the form based on the final layout.
5. Return the width and height.

Of course, the method by which the correct place for the node in the form is determined is where each of the possible layout algorithms needs to differ.

Our prototype uses a simple grid strategy as the core layout mechanism:

1. In regular layout, nodes are laid out in a column-based grid with configurable thresholds being used to limit the distance between elements vertically and horizontally.
2. In card layout, the nodes are divided into cards using logical groupings (such as container level elements or adjacent widgets in the XML node tree) and then the cards are arranged. Inside each card, the regular layout algorithm is applied (although recursion on container-level elements may lead to other layout methods appearing inside a card).

3. In minimal layout, a similar vertical grid approach is used as in regular layout but the minimal forms of the widgets is applied. In our prototype, widgets have two possible forms: a regular form and a minimal form designed specifically for the list-style minimal layout.

Figure 14 illustrates these three layout styles.

One of the underlying assumptions of this SUI rendering model implementation is that the multi-tiered approach to layout rendering using simple layout algorithms suited to different sized displays may produce effective results without the high CPU demands which might be exacted by a more complex, processing-intensive, “intelligent” layout algorithm.

A more processing-intensive algorithm will be problematic to implement in a small footprint and may not be functional on a small device with limited processing power. The multi-tiered approach adopted here requires little in the way of processing-intensive computation; if it produces results which are acceptable on a wide range of display sizes, this makes it a compelling option for small footprint devices.

Of course, this simpler approach has limitations including less control over the actual layout and aesthetics of the resulting form.

#### **4.4.3 Widget API**

Each widget is implemented as its own Flash movie and is compiled independent from the main rendering engine movie. Each widget must expose three methods which can be invoked by the rendering algorithms and the main rendering engine:

- `initialize`: this method takes as an argument the XML node object for the widget or form element or container being rendered (in the case of



a container, the entire tree beneath the node is included). In turn, it formats the widget in any available format so that sizes can be determined.

- **getSizes**: this method returns an array containing the width and height of the widget in each of its possible formats.
- **render**: this method takes as an argument which format of widget the rendering engine wishes to use and in turn displays that form of the widget.

The particular method by which the widget movie determines its available formats and sizes can be implemented in any way so long as these three methods are exposed.

At the present time each widget must implement all available rendering algorithms which means the addition of a rendering algorithm requires adding support to the widgets and recompiling them.

#### **4.4.4 Modularity: Using Multiple Flash Movies**

As mentioned earlier, the design of SUI makes use of multiple Flash movies: one for the main rendering engine plus one for each widget. This approach leverages Flash's ability to include compiled movies at run time. Using the `loadMovie` method in Flash, it is possible to load a movie into a container in another movie at runtime and then to access methods and properties of that movie from the parent movie. In this way the rendering engine is the parent movie and it creates a container for each widget and then loads the appropriate widgets' movies into each container based on the SUIML specification of the form being rendered.

Because each widget exposes the same set of methods to the rendering engine as described earlier in the discussion of the widget API this makes it possible for the rendering engine to load these widget movies at runtime and interact with them in a standard way without regard to the exact specifics of the internals of the widget's implementation. This also makes it possible for developers to extend or replace the widget movies while the rendering engine can still operate. In addition, new widgets can be added to the system. In both cases, the main rendering engine movie does not need to be recompiled. Essentially, this allows for the extensions of the SUIML language through the development of new widget movies. The SUI rendering engine relies on a simple naming convention: widget movies must possess the same name as the tag used to reference the widget in SUIML. Therefore, the movie which implements the `<text . . .>` tag in SUIML should be named `text.swf` (`.swf` is the extension used for compiled Flash movies).

To add a new widget and tag to SUIML is a simple task of choosing a name for the tag, creating a widget movie which implements the widget API and then using the tag in SUIML files. Because the main rendering engine and the widgets are compiled as separate Flash movie files it is not necessary to recompile the main rendering engine to accommodate a new widget—it simply needs to be added to the configuration file described in the next section; the main rendering engine uses the tag's name to locate the relevant widget's Flash movie file.

In addition, the widget movies are stored in the same directory as the rendering engine which allows the rendering engine movie to reference the widget movies using simple relative paths. In this way, the SUI rendering engine movie and the associated

widget movies can be placed on any HTTP server or tested from a local directory without any alteration of the movie files or special configuration required on the server.

#### 4.4.5 Flexibility and Configurability

Our SUI prototype implementation uses a three-tier rendering model with three different algorithms adopted as appropriate to the display space available. However, SUI in general is not restricted to three rendering algorithms.

The main rendering engine movie uses an XML-based configuration file to specify the available rendering algorithms and their order of preference. For instance, for our prototype implementation, this configuration code looks like:

```
<algorithms>
    <algorithm>regular</algorithm>
    <algorithm>card</algorithm>
    <algorithm>minimal</algorithm>
</algorithms>
```

Here, the order of preference for the rendering algorithms is “regular”, “card” and then “minimal”. These names should match the names of the rendering algorithm’s ActionScript files (such as `regular.as` or `minimal.as`) as well as the algorithm name which is specified in the code in the ActionScript file.

In addition, the configuration file contains a list of valid widget names. In order to use a new widget movie in a SUIML file, the widget should be added to the list in the configuration file:

```
<widgets>
```

```
<widget type="element">text</widget>

<widget type="container">box</widget>

<widget type="element">checkbox</widget>

etc.
```

```
</widgets>
```

Here, the order of widgets does not matter. Each widget has a type specified as either “container” or “element” which indicates if the widget is a container requiring recursive layout or if the element is an actual widget to be rendered itself.

Finally, the configuration file contains entries which specify both the maximum reduction scaling which the rendering engine can apply to a form, as well as the preferred distance between widgets horizontally and vertically (specified in pixels). these values are not device specific and will apply to any device; the default values shown here have been chosen based on informal experimentation with a range of values in varied windows sizes:

```
<scaling>90</scaling>

<spacing>

  <vertical>5</vertical>

  <horizontal>20</horizontal>

</spacing>
```

In addition, for testing purposes, the current prototype implementation places the main rendering engine inside a user interface which provides a simple form for controlling the dimensions of the display as well as constraints on the space between

elements (which is used by the regular layout algorithm) and the maximum reduction scaling percentage; this can allow testing of different configurations without editing configuration files. This interface is shown in Figure 16.

Changes to the configuration file do not require recompilation of the main Flash movie. If an algorithm is added, the present implementation requires the recompilation of individual widget files to add support for the new algorithm to the widgets.

**Figure 16**      **A form for controlling settings**

Canvas Size: X:  Y:  Divider Space: X:  Y:

Maximum Shrink:

## 4.5 Recursion

An important feature of the SUI implementation is the use of recursion. When certain container-level elements such as `box` or `tabPanel` are specified in a SUIML file, SUI implements them by simply including a new instance of the main rendering algorithm movie and passing the movie the appropriate branch of the XML node tree. This makes the main rendering algorithm movie a module which can be included in itself achieving recursion.

This strategy has been adopted to experiment with the results of using different rendering algorithms inside different container-level elements. When a container-level element occurs, a new instance of the main rendering algorithm movie is included and instead of checking the main display size, the parent provides a maximum display space available to the movie containing the child element.

This may result in layouts such as Figure 17. In Figure 17, the main part of the form has enough space to layout using a regular layout algorithm. However, this regular layout leaves insufficient space for the part of the form in the `box` container. The result is that it is rendered using a minimal layout algorithm.

If the form had more space, the `box` container might lay out using a regular layout algorithm as shown in Figure 18. Similarly, Figure 19 shows what happens when the display area becomes smaller and the entire movie switches to a minimal rendering style.

The SUIML code which generated this form is:

```
<text name="firstName" label="First Name" />
<text name="lastName" label="Last Name" />
<select name="age" label="Age" />
  <item name="birth_to_10" label="0-10"
value="0" />
  <item name="11_to_15" label="11-15" value="11">
```

```
    />
    <item name="16_to_21" label="16-21" value="16"
  />
  <item name="22_to_26" label="22-26" value="22"
  />
  <item name="27_to_35" label="27-35" value="27"
  />
  <item name="36_to_45" label="36-45" value="36"
  />
  <item name="46_to_55" label="46-55" value="46"
  />
  <item name="56_plus" label="56+" value="56" />
</select>
<box name="contactDetails" label="Contact
Details" border="true">
  <text name="phone" label="Phone Number">
  <text name="email" label="E-mail Address">
  <select name="country" label="Country">
    <item name="canada" label="Canada"
value="canada">
    <item name="usa" label="United States"
value="usa">
    <item name="mexico" label="Mexico"
value="mexico">
    <item name="other" label="Other"
value="other">
  </select>
</box>
<button name="submit" label="Submit">
```

**Figure 17** A mixture of regular and minimal layout

<b>First Name</b> Arman	<b>Contact Details</b>
<b>Last Name</b> Danesh	<b>Phone Number</b> 1-234-567-8901
<b>Age</b> 0-10 11-15 16-21 22-26 27-35	<b>E-mail Address</b> me@my.domain
	<b>Country</b> Canada ▶
	<b>Submit</b>

**Figure 18** More space leads to regular layout throughout

<b>First Name</b> Arman	<b>Contact Details</b>
<b>Last Name</b> Danesh	<b>Phone Number</b> 1-234-567-8901
<b>Age</b> 0-10 11-15 16-21 22-26 ---	<b>E-mail Address</b> me@my.domain
	<b>Country</b> Canada United States Mexico Other
	<b>Submit</b>

**Figure 19** Less space leads to minimal layout throughout

<b>First Name</b> Arman	<b>Contact Details</b>
<b>Last Name</b> Danesh	<b>Phone Nu...</b> 1-234-567...
<b>Age</b> 0-10 ▶	<b>E-mail Ad...</b> me@my.d...
	<b>Country</b> Canada ▶
	<b>Submit</b>



## CHAPTER FIVE: SAMPLE APPLICATIONS

To experiment with SUI and the rendering algorithms we have developed, two test forms have been created:

1. A user registration form: this form is representative of the common task of user registration on a Web site in which a user registers to create an account.
2. A store checkout form: this form is representative of tasks where the user must provide several distinct blocks of information including contact information, billing information, shipping information and payment information. This form is, necessarily, larger than a simple user registration form and can be used to push the layout algorithm further on small displays.

### *5.1 User Registration Form*

This form tests a common type of form found in Web applications: forms which require a small, self-contained set of information from the user and typically fits in a single screen.

The SUIML code used for this form is:

```
<text name="firstName" label="First Name" />
<text name="lastName" label="Last Name" />
<text name="email" label="E-mail Address" />
<select name="country" label="Country" />
  <item name="canada" label="Canada" value="ca"
/>
  <item name="unitedstates" label="United
```

```
States" value="us" />  
</select>
```

To test the effects of screen size, the form was rendered on progressively smaller displays of the following sizes:

- 640x480 pixels
- 320x240 pixels
- 240x320 pixels
- 150x150 pixels
- 90x90 pixels

These results are illustrated in Figure 20. Notice that the simplicity of the form allows it to render even on the smallest size display. In addition, Figure 20d shows the use of the card-based layout method. Here, there are four cards (for the first name, last name, e-mail address and country). When the user clicks on a card that card comes to the front and its contents are displayed; at other times, cards are minimized on the top and bottom edges of the display.

**Figure 20a** A user-registration form in SUIML (640x480 pixels)

First Name
Arman
Last Name
Danesh
E-mail Address
me@my.domain
Country
Canada
United States

**Figure 20b** A user-registration form in SUIML (320x240 pixels)

First Name	Country
Arman	Canada
Last Name	United States
Danesh	
E-mail Address	
me@my.domain	

**Figure 20c** A user-registration form in SUIML (240x320 pixels)

A user-registration form in SUIML (240x320 pixels). The form is a vertical stack of input fields. The first field is labeled "First Name" and contains the text "Arman". The second field is labeled "Last Name" and contains the text "Danesh". The third field is labeled "E-mail Address" and contains the text "me@my.domain". The fourth field is labeled "Country" and contains a list of options: "Canada" and "United States".

**Figure 20d** A user-registration form in SUIML (150x150 pixels); here a card-based layout is used and the e-mail address card is displayed in the center

A user-registration form in SUIML (150x150 pixels) using a card-based layout. The form is divided into three main sections. The top section contains two small input fields side-by-side, labeled "First Name" and "Last Name", with the text "Arman" and "Danesh" respectively. The middle section is a larger card containing the label "E-mail Address" and the text "me@my.domain". The bottom section contains a small input field labeled "Country" with the text "United States".

**Figure 20e** A user-registration form in SUIML (90x90 pixels)

A user-registration form in SUIML (90x90 pixels). The form is a vertical stack of input fields. The first field is labeled "First Name" and contains the text "Arman". The second field is labeled "Last Name" and contains the text "Danesh". The third field is labeled "E-mail Address" and contains the text "me@my.dom...". The fourth field is labeled "Country" and contains a list of options: "Canada" and "United States".

At all sizes, the form has rendered in a manner that keeps the form usable, even at the smallest size. Almost the entire form is visible which eliminates the need for scrolling. There are some noticeable limitations, including:

1. On the largest display, the form is anchored to the top-left corner of the display and large amounts of white space remain. Aesthetically it may be preferable to centre the form, change the dimensions of individual fields or otherwise alter the rendering; however, the SUI implementation does not account for any such aesthetics.
2. The 150 x 150 pixel display shows that the best decision is not always made. Here, a card layout is used which requires the user to open each card to access the individual widgets in the form. However, if the rendering engine had continued down to the minimal algorithm and used it all four fields would have been visible on the screen at one time.

At the same time, however, the intermediate sizes (240x320 and 320x240) allow the form to be rendered quite well: the available white space is better used than at 640x480 without excessive white space and all widgets are rendered with the larger, more legible regular widgets.

## ***5.2 Store Checkout Form***

This form is much larger and more complex than the user registration form. It also invokes a wider range of user interface elements and uses recursive layout through the use of the `box` tag to contain each separate set of information being sought from the user.

The form requires four distinct sets of information from the user:

1. Personal contact information such as phone number, fax number and e-mail address.
2. Billing address information

3. Shipping address information
4. Payment information including credit card data

The SUIML code used for this form is:

```
<box name="personal" label="Personal Data"
border="true">
  <text name="firstName" label="First Name" />
  <text name="lastName" label="Last Name" />
  <text name="email" label="E-mail Address" />
</box>

<box name="billing" label="Billing Address"
border="true">
  <text name="billingaddress" label="Address"
lines="5" />
  <text name="billingcity" label="City" />
  <select name=" billing province"
label="Province">
    <item name="billingbc" label="British
Columbia" value="bc" />
    <item name="billingab" label="Alberta"
value="ab" />
    <item name="billingsk" label="Saskatchewan"
value="sk" />
    <item name="billingmb" label="Manitoba"
value="mb" />
    <item name="billingon" label="Ontario"
value="on" />
    <item name="billingqc" label="Quebec"
value="qc" />
    <item name="billingnb" label="New
Brunswick" value="nb" />
    <item name="billingsns" label="Nova Scotia"
value="ns" />
    <item name="billingnf" label="Newfoundland"
value="nf" />
    <item name="billingpe" label="Prince Edward
Island" value="pe" />
    <item name="billingnu" label="Nunavut"
value="nu" />
    <item name="billingnw" label="Northwest
Territories" value="nw" />
    <item name="billingyk" label="Yukon
Territories" value="yk" />
  </select>
```

```

    <text name="postcode" label="Postal Code" />
</box>

<box name="shipping" label="Shipping Address"
border="true">
    <text name="shippingaddress" label="Address"
lines="5" />
    <text name="shippingcity" label="City" />
    <select name="shippingprovince"
label="Province">
        <item name="shippingbc" label="British
Columbia" value="bc" />
        <item name="shippingab" label="Alberta"
value="ab" />
        <item name="shippingsk"
label="Saskatchewan" value="sk" />
        <item name="shippingmb" label="Manitoba"
value="mb" />
        <item name="shippingon" label="Ontario"
value="on" />
        <item name="shippingqc" label="Quebec"
value="qc" />
        <item name="shippingnb" label="New
Brunswick" value="nb" />
        <item name="shippingns" label="Nova Scotia"
value="ns" />
        <item name="shippingnf"
label="Newfoundland" value="nf" />
        <item name="shippingpe" label="Prince
Edward Island" value="pe" />
        <item name="shippingnu" label="Nunavut"
value="nu" />
        <item name="shippingnw" label="Northwest
Territories" value="nw" />
        <item name="shippingyk" label="Yukon
Territories" value="yk" />
    </select>
    <text name="shippingpostcode" label="Postal
Code" />
</box>

<box name="payment" label="Payment Information"
border="true">
    <radioGroup name="cardType" label="Credit
Card">
        <radio name="visa" label="Visa"
value="visa">
        <radio name="mastercard" label="MasterCard"

```

```
value="mastercard">
  </radioGroup>
  <text name="ccnumber" label="Card Number">
  <text name="ccexpiry" label="Expiry Date">
</box>
```

To test the affects of screen size, the form was rendered on progressively smaller displays of the following sizes:

- 640x480 pixels
- 320x240 pixels
- 240x320 pixels
- 150x150 pixels
- 90x90 pixels

These results are illustrated in Figure 21. Notice several interesting results including the intermediate stage where some of the boxes use a minimal layout while others use a regular layout. Also notice that at 90 x 90 pixels becomes so small that limited context information can be made available to the user.

This form illustrates that as forms get more complex, more issues are introduced in the layout and the algorithms at play have more trouble achieving a clean, simple, attractive layout of the form than was seen with the previous, simpler form.

Here, at 640 x480, all fields fit relatively well; the form fills most of the available space in even columns which require no scrolling; all widgets are of the larger, regular variety.



From 240x320 pixels and smaller, all layout is done with the minimal style. The card layout is never adopted because in the large sizes, it isn't necessary and the smaller 90x90 pixel display is simply too small to render the cards.

The most problematic size is the 320x240 pixel layout where the rendering engine sees that the regular layout style can be used for the widgets in the first box but subsequently, the recursive process leads to a minimal layout for the widgets inside the other three boxes. The result is a form with two styles of widgets which look different and behave different.

**Figure 21a** A Store Checkout form in SUIML (640x480 pixels); image is scaled to fit on page. All fields fit relatively well and no scrolling is required.

Personal Data	Billing Address	Shipping Address	Payment Information
First Name <input type="text"/>	Address <input type="text"/>	Address <input type="text"/>	Credit Card <input type="checkbox"/> Visa <input type="checkbox"/> MasterCard
Last Name <input type="text"/>			
E-mail Address <input type="text"/>	City <input type="text"/>	City <input type="text"/>	Card Number <input type="text"/>
	Province British Columbt ▲ Alberta Saskatchewan Manitoba ▼	Province British Columbt ▲ Alberta Saskatchewan Manitoba ▼	Expiry Date <input type="text"/>
	Postal Code me@my.domain	Postal Code me@my.domain	

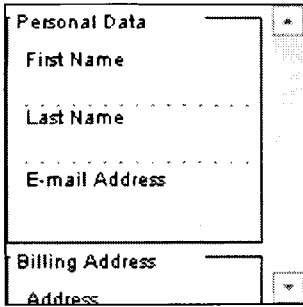
**Figure 21b** A Store Checkout form in SUIML (320x240 pixels). At this size, a combination of widget styles appear in the different box elements.

<b>Personal Data</b> First Name <input type="text"/> Last Name <input type="text"/> E-mail A... <input type="text"/>	<b>Billing Address</b> Address <input type="text"/> City <input type="text"/> Province <input type="text"/> British Co... ▶ Postal Code <input type="text"/>	<b>Shipping A...</b> Address <input type="text"/> City <input type="text"/> Province <input type="text"/> British Co... ▶ Postal Code <input type="text"/>
<b>Payment Info...</b> Credit Card <input type="text"/> Msa ▶ Card Nu... <input type="text"/> Expiry Date <input type="text"/>		

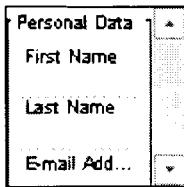
**Figure 21c** A Store Checkout form in SUIML (240x320 pixels). All widgets are rendered in the minimal style.

<b>Personal Data</b> First Name <input type="text"/> Last Name <input type="text"/> E-mail Add... <input type="text"/>	<b>Shipping A...</b> Address <input type="text"/> City <input type="text"/> Province <input type="text"/> British Co... ▶ Postal Code <input type="text"/>
<b>Billing Address</b> Address <input type="text"/> City <input type="text"/> Province <input type="text"/> British Co... ▶ Postal Code <input type="text"/>	<b>Payment Info...</b> Credit Card <input type="text"/> Visa ▶ Card Number <input type="text"/> Expiry Date <input type="text"/>

**Figure 21d** A Store Checkout form in SUIML (150x150 pixels). At this size, minimal widgets are used but scrolling becomes necessary.



**Figure 21e** A Store Checkout form in SUIML (90x90 pixels). At extremely small sizes, there is no choice but to use minimal widgets.



## CHAPTER SIX: DISCUSSION

The implementation and testing of SUI for this thesis highlights several key points:

- Flash provides a robust framework for developing consistent user interfaces across client platforms.
- The simple three-tiered layout model used in the current SUI implementation is limited. SUI relies on levels of simple algorithms rather than complex, highly-intelligent algorithms to achieve display-appropriate form rendering. This led to less-than-ideal results in borderline cases.
- Moving the rendering logic out to the client using Flash can impose performance limitations since not all clients have equal levels of processing power.
- The recursion model created form layouts which may cause confusion in users because of the varied layout models used in rendering a single form.
- Using automatic layout means there is a sacrifice of standardization and consistency.

### ***6.1 Flash as a User Interface Framework***

Regardless of the size of the target client's display, the form widgets displayed consistently on different platforms; for example, a button rendered in the SUI minimal style appears the same on any platform as will a text box rendered in the normal style.

Testing across Windows, Linux and the latest incarnation of Microsoft's handheld

operating system, Windows Mobile 2003 shows that Flash widgets display consistently across these platforms.

The critical difference is that release schedules differ on different platforms; Windows and Macintosh OS X users will receive new versions of the Flash Player before users on other platforms which means that applications which implement features which require the newest player will not immediately run on all platforms and must wait for full cross-platform compatibility for the staged release of the new player on other platforms. Still, the assurance of consistency of appearance and the ability to run a single compiled Flash movie file on all platforms combined with user interface components and a component development model in Flash MX makes Flash a strong candidate for cross-platform user-interface implementations.

Implementing SUI provided evidence of this utility for user interface development. No specific consideration had to be given to individual client platforms during development; to the contrary, the client platform was completely ignored in the development of the SUI implementation. Still, the application ran consistently in Flash Player 7 on the platforms for which it was available.

## ***6.2 Simple vs. Intelligent Algorithms***

The architecture of SUI as implemented in this work is to use relatively simple layout algorithms combined through a rendering engine which attempts to use the algorithms in a defined sequence to find a viable layout. Therefore, instead of relying on the implementation of a single, complex rendering algorithm, SUI opts to use simple algorithms but then to provide the rendering engine with several alternative algorithms

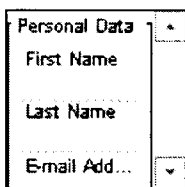
with the expectation that this will produce usable results with a simpler, more manageable implementation.

The results of the sample application presented earlier show that this option produces layouts for simple and complex forms on various sized-displays which are functional and can be used. This is done by implementing simplistic layout algorithms which use a grid and a vertical, column-based strategy to position form elements.

### 6.2.1 Problems with Small Displays

In border line cases, the simple algorithm strategy has led to less than ideal results. In particular, at extremely small screen sizes, even small forms render in a format which may not be entirely usable for all users. Consider Figure 22 where the store checkout form is rendered on a small 90 x 90 pixel display:

**Figure 22**      **The store checkout form on a 90 x 90 pixel display**



Personal Data	*
First Name	
Last Name	
Email Add...	

While this form is legible, there are some issues to consider:

- The fields have been scaled to 90% of their normal size making the type smaller and harder to read.
- Less than 25% of the form is visible at any one time which makes it hard for a user to keep their context clear in the form as well as to know how much of the form they have completed.

- The introduction of the scrollbar itself consumes display space which could have been used, for instance, for wider text fields or to avoid scaling of the fields.

### **6.2.2 Problems with Large Displays**

Other problems arise from large forms with lots of fields rendered on reasonably-sized displays; these will often have several columns of form fields and the ordering and column breaks used will not reflect the purpose or intent of the form's creator as in Figure 23 showing the shopping cart form on a large 1024 x 768 pixel display.

Figure 23

The shopping cart form on a large 1024 x 768 pixel display (rotated 90 degrees and scaled to 75% of actual size to fit page)

The form is organized into four distinct sections, each with a title and several input fields:

- Personal Data:** Includes fields for First Name, Last Name, and E-mail Address.
- Shipping Address:** Includes fields for Address, City, Province (with a dropdown menu listing British Columbia, Alberta, Saskatchewan, and Manitoba), and E-mail Address (with a placeholder 'me@my.domain').
- Billing Address:** Includes fields for Address, City, Province (with a dropdown menu listing British Columbia, Alberta, Saskatchewan, and Manitoba), and E-mail Address (with a placeholder 'me@my.domain').
- Payment Information:** Includes a Credit Card field (with a dropdown menu listing Visa and MasterCard), a Card Number field, and an Expiry Date field.



In this situation, the layout has several issues, including:

- Screen space is not well-utilized. Instead of spreading the form more horizontally across the available width the entire form hugs the left side of the display.
- The form boxes are narrower than they need to be because this is the default width widgets use when possible.

As a general rule, SUI will automatically render a form but it will not necessarily deliver an ideal or optimal layout; the resulting layout will work and be usable on the device in question but it may not be attractive or the easiest layout for user interaction.

### ***6.3 Client Performance***

While Flash can provide an assurance of consistency of implementation and behaviour across platforms, it cannot provide guarantees of performance. The computing resources available in a device do impact the performance of individual Flash movie files running on different hardware.

To benchmark this, we utilized an ability of the Flash development environment to generate tracing output while running a compiled Flash movie within the Flash MX 2004 development environment. This allowed the date stamp to be output as tracing output when rendering started and when it completed to gain a sense of the time required to perform rendering on different systems. Because the Flash MX 2004 development tool cannot run on handheld devices and requires the full Windows operating system, this tracing was not conducted on a handheld device. However, by using desktop computers with a range of CPU power, it is possible to plot the approximate performance issues which arise as the rendering engine is executed on hardware with less resources.

A simple comparison of two systems highlights the fact that hardware does impact the speed of rendering. The hardware used for testing was:

- A 1.2 GHz AMD Duron desktop computer with 1 GB of RAM
- A 600 MHz Celeron notebook computer with 256 MB RAM

The results for our two sample applications are as follows:

**Table 1 Performance results for test applications**

	<b>User Registration Form</b>	<b>Store Checkout Form</b>
<b>1.2 GHz Desktop</b>	600x480: 0.2 seconds	640x480: 0.6 seconds
<b>1 GB RAM</b>	320x240: 0.3 seconds	320x240: 0.9 seconds
<b>600 MHZ Notebook</b>	640x480: 0.5 seconds	640x480: 0.8 seconds
<b>256 MB RAM</b>	320x240: 0.8 seconds	320x240: 1.5 seconds

Three main trends are suggested by these rendering-time results:

1. CPU speed and RAM, as the two key hardware resources affecting performance, do have an impact on rendering time.
2. The complexity of the SUIML form specification impacts the rendering time. On both platforms, rendering time increased with the more complex store checkout form.
3. As screen size decreases, rendering time increases; this is most notably true for the more complex store checkout form. The reason for this is that as a form must be rendered on increasingly smaller displays, it is increasingly likely that the rendering engine will have to try more of the

rendering algorithms. For instance, on a large display, rendering might start with the regular rendering algorithm and end there but on a small display the same form might be rendering first with the regular algorithm, then with the card-based algorithm and finally with the minimal algorithm before producing a final, usable layout. This latter process, of course, requires more time to complete.

These trends suggest that there may be performance issues with more complex forms on smaller devices. As a device gets smaller, available CPU power and RAM decreases while the screen size also decreases. This means the most complex rendering process will be executed on systems with the least resources.

#### ***6.4 The Benefits and Drawbacks of Recursion***

The recursive nature of the layout model used in our implementation of the SUI rendering engine appears to offer benefits as well as drawbacks. The primary benefit of the method is in its ability to provide the standard widgets to the user as often as possible. The regular rendering algorithm uses standard form widgets which will be familiar to users with prior experience with graphical user interfaces.

Consider the case of the sample store checkout form; At 320 x 240 pixels, the layout still retains some regularly-formatted widgets in the "Personal Data" box as illustrated in Figure 24.

**Figure 24** The store checkout form at 320 x 240 pixels

The diagram illustrates a store checkout form layout at 320 x 240 pixels. It is organized into four distinct sections, each enclosed in a box with a title:

- Personal Data:** Contains input fields for First Name, Last Name, and E-mail Address.
- Billing Address:** Contains input fields for Address, City, Province (with a dropdown menu showing 'British Co...'), and E-mail Address.
- Shipping Address:** Contains input fields for Address, City, Province (with a dropdown menu showing 'British Co...'), and E-mail Address.
- Payment Information:** Contains input fields for Credit Card (with a dropdown menu showing 'Visa'), Card Number, and Expiry Date.

Without recursion, three options exist:

- The "Personal Data" box is rendered with minimal widgets in it.
- The entire layout is done with a card-style layout and individual cards are likely rendered with minimal widgets in them.
- Regular widgets are used throughout but are shrunk to 90% of their original size or possibly even smaller depending on configuration settings.

Informal visual inspection of the various layouts from the sample application which used recursion highlight a fundamental problem with the recursion as presently implemented: a lack of consistency. Consider the same store checkout form at 320 x 240 pixels illustrated in Figure 24.

Here we can see three types of layout in use:

1. The personal data box uses regular layout within it.
2. The two address boxes uses minimal layout within them.

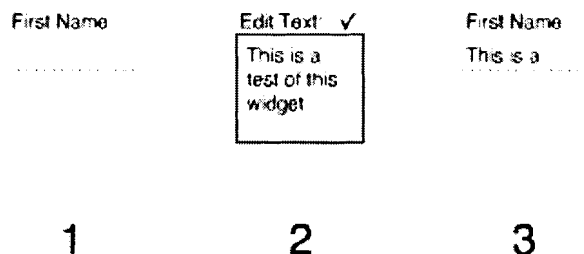
3. The payment details box uses minimal layout within it but this is then scaled to 90% of the standard size and a scroll bar is introduced.

This mixture is visually disconcerting: the various box level elements look considerably different internally. Plus the same widgets show distinctly different behaviour. For instance, a regular text field works like familiar text fields from HTML forms. The user simply places the cursor in the text field and types. In the minimal layout modality, text fields are designed differently. Minimal text fields work as follows:

1. By default, minimal text fields display a single line of text underneath the label for the field.
2. When the user clicks in the text field, it expands to a larger box in which the user can type; this box will cover any widgets which might be below.
3. When the user finishes typing, they can click on the icon in the top right; the width shrinks back to a single line with the first few words of text displayed.

These steps are illustrated in Figure 25.

**Figure 25**      **Operation of a minimal text widget**



The minimal text field requires a considerably different user interaction than a normal text field and raises the question of impact on the user and the usability of the

form when recursion leads to mixed widget styles. The problem could become especially acute if multiple algorithms (more than two), with associated widget styles, are used in the same form.

### ***6.5 Standardization and Consistency***

The current SUI implementation has some implications with respect to standardization and consistency. Because SUI uses a form of automatic layout there are no guarantees about the layout and appearance of forms. Not only can the layout of a form differ widely from device to device but the user cannot assume where buttons, fields or other form elements will appear each time they access a form.

In addition, it is important to note that the use of Flash means that widget appearance and behaviour will be consistent across devices but may differ from standard widget appearance and behaviour on a given device or operating system. Flash's default widgets are not intended to match the widget appearance and behaviour of a host operating system's widgets.

### ***6.6 Implications for Developer and Designers***

One of the main motivations of SUI is to simplify the job of implementing interactive forms which work across multiple devices and client platforms. As a result, SUI has implications for Web application developers who can realize a simpler, more streamlined development process for these forms because only one set of source code is required to implement and deploy a given form. However, this simplified development process comes with a trade-off: it limits the ability of the designer to control the design and appearance of a form. The automatic layout used by SUI means the designer effectively has no control of the placement of widgets. In addition, the approach of

multiple rendering algorithms means that widgets have multiple forms so designers also have no guarantee of which style of widget will be used. This doesn't mean designers could not exercise any control. The modular nature of SUI means that designers can develop their own widgets with unique visual appearance and these can easily be deployed with small changes to the SUI configuration file. In addition, Flash supports limited use of CSS for styling widget components. Although SUI does not employ this capability, it would be possible to extend SUI to support this feature. Nonetheless, to use SUI does mean sacrificing the fine-grained visual control designers are used to with today's Web applications.

## **CHAPTER SEVEN: CONCLUSIONS AND FUTURE WORK**

Several contributions emerge from this current work in Scalable User Interfaces for the Web:

- Flash provides a compelling platform for cross-device Web form delivery.
- It is possible to support several display form factors, and in particular smaller displays, from a single specification of a form.
- A recursive rendering strategy provides a way to use simple automated rendering algorithms while allowing a form to scale across a wide range of display sizes.

### ***7.1 Using Flash for Cross-Device Form Delivery***

Previous work in the area of cross-device delivery of Web user interfaces has not considered the use of Flash as a suitable platform. Since the release of Flash MX, there has been discussion in the industry, led by Macromedia, of Flash's role in the delivery of cross-platform interactive content.

However, this industry discussion has not been supported by reports in the literature. The present implementation of Scalable User Interfaces, which has been developed using Flash, shows that Flash can successfully be used to delivery single executable applications and user interfaces to multiple devices and platforms.

This project demonstrates as well that, as Macromedia claims, Flash runs in a consistent fashion across multiple platforms and devices.



Finally, Flash's integrated XML support made it easy for the Scalable User Interfaces implementation to work with form specifications written in an XML-based language; this suggests Flash may be suited to other applications which rely on XML-based data.

### **7.1.1 Implications for SVG and Other Platforms**

Our success using Flash as a platform for deploying a single executable SUI rendering engine which runs on all platforms also provides insights which are relevant to other similar technologies such as SVG. SVG offers several key features in common with Flash which were important in the implementation of the SUI prototype:

- Cross-platform consistency
- Scalability to small devices
- An ECMAScript scripting environment

Our success using Flash to create a single executable rendering application which runs on all supported platforms may translate to SVG. Not only is it possible to consider implementing SUI in SVG, these similarities also indicates that platforms such as SVG lend themselves to creating these types of executable applications which run on multiple platforms without porting, adaptation or recompilation.

### ***7.2 Automated Form Rendering for Small Displays***

A key feature of the Scaleable User Interfaces is that it provides for automated form rendering so that forms are not only usable on standard desktop-sized displays but can similarly be rendering in an automated fashion on a variety of small displays, particularly in portable, handheld devices.

Scalable User Interfaces is a mechanism which can, in fact, render a single form specification on multiple displays of different sizes without requiring the form developer to specify separate form definitions for different displays or to provide hinting to the rendering engine to handle different target devices.

Qualitative analysis of the results shows that results may not be ideal on every display size but that the forms are rendered and can be used on most display sizes in common use today.

### ***7.3 Using Recursion and Simple Algorithms for Form Rendering***

Rather than creating a single monolithic rendering algorithm which attempts to intelligently decide how to render a given form on a specific display, Scalable User Interfaces adopts a modular, tiered strategy which differs from previous work in the area of form rendering for small displays.

Scalable User Interfaces shows that the recursive, tiered layering of multiple, simple, rendering algorithms provides a scalable way to render forms across multiple display sizes including smaller, hand-held displays. At the same time, recursion has limitations notably in the area of consistency: a form may be rendered using widgets of different styles.

#### **7.3.1 The Importance of Simple Algorithms**

SUI's use of a sequence of simple algorithms may have broader implications than simply for the layout of Web forms. In other cases where a single complex algorithm attempts to account for a set of different conditions, it may prove easier to develop a set of simpler algorithms which, used in the correct sequence, provide adequate results without the more daunting development task of a single, monolithic algorithm.

### **7.3.2 The Relevance of Recursion**

SUI illustrates the applicability of a recursive strategy to the work of rendering forms. This application of recursion is not limited to SUI or a Flash-based rendering engine. The lessons learned in SUI about recursion with simple algorithms could be applied to the layout of standard HTML forms in regular Web browsers, could be adopted by clients which render XForms or could be used even more generally in developing tools for rendering Web pages on small devices.

### ***7.4 Future Work***

The work on SUI described in this thesis has provided a proof-of-concept implementation of the SUI model in Flash and has implemented and informally assessed the layout of two test applications to assess the mode.

This work, however, suggests several areas of future work:

- Testing of alternate rendering algorithms.
- Implementation of a complete widget set
- Improvements to the efficiency of algorithms
- Testing of the system with end users

#### **7.4.1 Test Alternate Rendering Algorithms**

The design of SUI allows for the implementation of alternate rendering algorithms. The present implementation implements three algorithms: a regular algorithm, a card-based algorithm and a minimal algorithm.

The actual design of these individual algorithms is relatively simple. None of the algorithms involve complex rules or processes for deciding how to layout widgets but

adopt a simple, vertical-oriented grid approach. Analysis shows that in borderline cases, the layouts that are produced are less-than-ideal. However, the design of more complex, more intelligent individual rendering algorithms combined with SUI's tiered rendering engine with options for recursive layout may produce interesting new results both in terms of improving the use of screen real estate, improving the viability of the recursive layout or providing a more consistent, usable user experience.

#### **7.4.2 Implement the Complete Widget Set**

The current implementation has implemented a subset of the widget set specified in SUIML. This limited implementation provides sufficient widgets to implement viable test applications. However, the SUIML widget set includes more sophisticated widgets which would offer interesting possibilities for implementing more sophisticated, powerful user interfaces.

Future work to implement the complete widget set would allow further testing of the types of layouts produced by SUI, the efficiency of layout and the viability of the recursive layout used by the SUI rendering engine.

#### **7.4.3 Improve Efficiency of Algorithms for Small, CPU-Limited Devices**

Our test results on two different platforms show that there can be noticeable differences in rendering time on different hardware and that on extremely small, resource-limited devices the reduced performance may become noticeable by users as the form complexity increases.

Attention to the efficiency of the rendering algorithms and designing more efficient algorithms would provide a way to address these possible limitations.

It may also be possible to improve efficiency by adjusting the ordering of the algorithms. In the current implementation, rendering proceeds by starting with the algorithm suited to larger displays and moving progressively towards the algorithm primarily aimed at smaller displays until a suitable layout is found. However, devices with larger displays tend to have more processing power than smaller, handheld devices. But, this strategy means that smaller devices will generally have to work through more of the algorithms to reach a usable layout. By reversing the order in which algorithms are attempted and starting with the algorithm intended for smaller displays it should be possible to minimize the number of rendering attempts smaller, resource-limited devices must make and, thus, improve rendering time on these devices.

#### **7.4.4 Conduct User Testing**

Our current work has involved the development of a proof-of-concept implementation of a SUI system in Flash and informal assessment by the developer with two test applications. This assessment of the resulting layouts and the effectiveness of the layouts and, in particular, the recursive nature of the layout, has been conducted on an informal, qualitative basis.

To better assess the effectiveness of the form layouts and the implications of recursive rendering, formal user testing should be conducted.

## REFERENCES

1. Abrams, M. Device-Independent Authoring with UIML. W3C Workshop on Web Device Independent Authoring, Bristol. October 2000.
2. Adams, P. Intro to HDML. WebMonkey, 9 May 2001.  
<http://hotwired.lycos.com/webmonkey/99/48/index3a.html?tw=design>.
3. Ali, M. F., and Abrams, M. Simplifying Construction of Multi-Platform User Interfaces Using UIML. UIML Europe 2001, March, 2001.
4. Allaire, Jeremy. Macromedia Flash MX: A Next-Generation Rich Client. March 2002. <http://www.macromedia.com/desdev/mx/flash/whitepapers/richclient.pdf>.
5. Allaire, Jeremy. Macromedia MX: Components and Web Services. April 2002.  
[http://www.macromedia.com/desdev/mx/coldfusion/whitepapers/components\\_ws.pdf](http://www.macromedia.com/desdev/mx/coldfusion/whitepapers/components_ws.pdf).
6. Amazon.com. Amazon.com Web Services 2.0.  
<http://www.amazon.com/webservices>.
7. Apache Axis Development Team. Apache Axis. <http://ws.apache.org/axis/>.
8. Aslam, S. WEB-Based Query Processing in a Database Course Project. Proceedings of the 29<sup>th</sup> SIGCSE Technical Symposium on Computer Science Education, Atlanta, 1998, pp. 297-301.
9. Auld, C., Spencer, P., Rafter, J., James, J., Addey, D., Gauti, O., Fudmundsson, O. G., Kent, A., Schiell, A., and Surguy, I. *Practical XML for the Web*, Glasshaus, October 2002.
10. Barth, P. S. An Object-Oriented Approach to Graphical Interfaces. ACM Transactions on Graphics, Volume 5, Number 2, April 1986.
11. Baudisch, P., Good, N., Bellotti, V., and Schraedley, P. Keeping Things in Context: A Comparative Evaluation of Focus Plus Context Screens, Overviews, and Zooming. Proceedings of the 2002 SIGCHI Conference on Human Factors in Computing Systems, Minneapolis, 2002, pp. 259-266.
12. BEA Systems, Inc. <http://www.bea.com/>.
13. Bederson, B. B., and Hollan, J. D. Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. Proceedings of the 7<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology, Marina del Rey, 1994.

14. Berners-Lee, Tim. Information Management: A Proposal. March 1989, May 1990. <http://www.w3.org/History/1989/proposal.html>.
15. Bickmore, T. W. and Schilit, B. N. Digester: Device-Independent Access to the World Wide Web. Selected Papers from the Sixth International Conference on the World Wide Web, Santa Clara, 1997.
16. Bjork, S., Holmquist, L. E., Redstrom, J., Bretan, I., Danielsson, R., Karlgren, J., and Franzen, K. WEST: A Web Browser for Small Terminals. Proceedings of the 12<sup>th</sup> Annual ACM SIGGRAPH Symposium on User Interface Software and Technology, Asheville, 1999, pp. 187-196.
17. Bleser, T., and Foley, J. D. Towards Specifying and Evaluating the Human Factors of User-Computer Interfaces. Proceedings of the 1<sup>st</sup> Major Conference on Human Factors in Computing Systems, Gaithersburg, 1982.
18. Bojanic, P. The Joy of XUL. February 2002. <http://www.mozilla.org/projects/xul/joy-of-xul.html>.
19. Brabrand, C., Moller, A., and Schwartzbach, M. I. The <bigwig> Project. ACM Transactions on Internet Technology, Volume 2, Number 2, May 2002, pp. 79-114.
20. Britt, J. On-Request Conversion of HTML to WML. TopXML. <http://www.vbxml.com/WAP/html2wap.asp>.
21. Browning, P., and Lowndes, P. JISC TechWatch Report: Content Management Systems. September 2001. [http://www.jisc.ac.uk/uploaded\\_documents/tsw\\_01-02.pdf](http://www.jisc.ac.uk/uploaded_documents/tsw_01-02.pdf).
22. Buckingham, Simon. What is General Packet Radio Service. 2000. <http://www.gsmworld.com/technology/gprs/intro.shtml>.
23. Buyukkokten, O., Garcia-Molina, H., Paepcke, A., and Winograd, T. Power Browser: Efficient Web Browsing for PDAs. Proceedings of the SIGCHI Conference on Human Factors and Computing Systems, The Hague, 2000.
24. Buyukkokten, O., Kaljuvee, O., Garcia-Molina, H., Paepcke, A., and Winograd, T. Efficient Web Browsing on Handheld Devices Using Page and Form Summarization. ACM Transactions on Information Systems, Volume 20, Issue 1, January 2002, pp. 82-115.
25. Cerf, Vincent G. The Invisible Internet: Beyond the Post-PC Internet. Communications of the ACMU, Volume 44, Number 9, September 2001, pp. 34-40.

26. Champion, M, Ferris, C., Newcomer, E., and Orchard, D., eds. Web Services Architecture. W3C Working Draft, 14 November 2002. <http://www.w3.org/TR/ws-arch/>.
27. Coder, L. Dynamic HTML Conversion to WML. <http://html2wml.sourceforge.net/>.
28. ComputerScope, Ltd. META Group: Web Content Management Market to Grow. August 2002. [http://www.nua.ie/surveys/index.cgi?f=VS&art\\_id=905358280&rel=true](http://www.nua.ie/surveys/index.cgi?f=VS&art_id=905358280&rel=true).
29. Cross-Browser.com. <http://www.cross-browser.com/>.
30. Crucial Technology. <http://www.crucial.com/>.
31. Danesh, Arman. *Mastering ColdFusion 5*, Sybex, 2001.
32. de Baar, D. J. M. J., Foley, J., and Mullet, K. E. Coupling Application Design and User Interface Design. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Monterey, 1992.
33. Deakin, N. XUL Tutorial. February 2003. <http://www.xulplanet.com/tutorials/xultu/>.
34. DNA Data Bank of Japan. XML Central of DDBJ. <http://xml.nig.ac.jp/index.html>.
35. European Computer Manufacturers Association. ECMAScript Language Specification. December, 1999. <http://ecma-international.org/publications/files/ecma-st/Ecma-262.pdf>.
36. Evolt.org. Your Clients Need a Contact Management System. March 2001. <http://www.evolt.org/article/rating/20/5127/>.
37. Exchange User Education. Microsoft Outlook Web Access in Microsoft Exchange Server 2000. Exchange Core Documentation, May 2002. [http://download.microsoft.com/download/exchplatinumbeta/e2kowa/1.0/WIN98/MeXP/EN-US/e2k\\_owa\\_pdf.exe](http://download.microsoft.com/download/exchplatinumbeta/e2kowa/1.0/WIN98/MeXP/EN-US/e2k_owa_pdf.exe).
38. Foley, J. Gibbs, C., Kim, W. C., and Kovacevic, S. A Knowledge-Based User Interface Management System. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Washington, D.C., 1992.
39. Furnas, G. W. Generalized Fisheye Views. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, April 1986, pp. 16-23.
40. Garmo, R., Hagar, B., and Wojtowicz, M. Intranets. July 2001. <http://www-personal.umd.umich.edu/~rgarmo/>.



41. Gerken, T., and Ratschiller, T. *Web Application Development with PHP*, New Riders Publishing, 2000.
42. Green, Mark. A Survey of Three Dialogue Models. *Transactions on Graphics*, Volume 5, Number 3, July 1986.
43. Grundy, J., Wang, X., and Hosking, J. Building Multi-Device, Component-Based, Thin-Client Groupware: Issues and Experiences. *Third Australian Conference on User Interfaces*, Melbourne, 2002, pp. 71-80.
44. Guldman, A. Building Rich Internet Applications with Macromedia Flash MX and ColdFusion MX. May 2002.  
[http://www.macromedia.com/resources/business/sample\\_apps/customer/customer\\_app.pdf](http://www.macromedia.com/resources/business/sample_apps/customer/customer_app.pdf).
45. Handspring. <http://www.handspring.com/>.
46. Harmonia, Inc. The UIML Vision. February 2000.  
<http://www.harmonia.com/resources/papers/whitepapers/UIMLVisionWhitePaperV5.pdf>.
47. Hayes, P. J., Szekely, P. A., and Lerner, R. A. Design Alternatives for User Interface Management Systems Based on Experience with Cousin. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, San Francisco, 1985.
48. Hewlett-Packard Company. <http://www.hp.com/>.
49. Holland, J. An Introduction to PHP3. *Linux Journal*, Volume 2000, Issue 73, May 2000.
50. Hudson, S. Graphical Specification of Flexible User Interface Displays. *Proceedings of the 2<sup>nd</sup> Annual ACM SIGGRAPH Symposium on User Interface Software and Technology*, Williamsburg, November, 1989.
51. Hyatt, D., ed. XML User Interface Language (XUL) 1.0. 2001.  
<http://www.mozilla.org/projects/xul/xul.html>.
52. Jacob, R. J. K. Using Formal Specifications in the Design of a Human-Computer Interface. *Communications of the ACM*, Volume 26, Number 4, April 1983.
53. Janssen, C., Weisbecker, A., and Ziegler, J. Generating User Interfaces from Data Models and Dialogue Net Specifications. *Proceedings of the 1993 SIGCHI Conference on Human Factors in Computing Systems*, Amsterdam, 1993.
54. Kamada, T. Compact HTML for Small Information Appliances, W3C Note, 9 February 1998. <http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/>.

55. Kaptelinin, V. A Comparison of Four Navigation Techniques in a 2D Browsing Task. Proceedings of the SIGCHI Conference on Human factors in Computing Systems, Denver, 1995, pp. 282-283.
56. Kim, W. C. and Foley, J. D. DON: User Interface Presentation Design Assistant. Proceedings of the 3<sup>rd</sup> Annual ACM SIGGRAPH Symposium on User Interface Software Technology, Snowbird, 1990.
57. Labrinidis, A., and Roussopoulos, N. Generating Dynamic Content at Database-Backed Web Servers: cgi-bin vs. mod\_perl. ACM SIGMOD Record, Volume 29, Issue 1, March 2000, pp. 26-31.
58. Lemay, L. and Danesh, A. Teach Yourself Web Publishing with HTML 4 in a Week, Sams.net, 1997.
59. Livingston, D. Essential CSS and DHTML for Web Professionals, Prentice Hall PTR, July 2001.
60. Lok, S., Feiner, S. K., Chiong, W. M., and Hirsch, Y. J. A Graphical User Interface Toolkit Approach to Thin-Client Computing. Proceedings of the 11<sup>th</sup> International Conference on World Wide Web, Honolulu 2002, pp. 718-722.
61. Macromedia. Developing Rich Internet Applications with Macromedia MX. April 2002.  
[http://www.macromedia.com/desdev/mx/studio/whitepapers/rich\\_internet\\_apps.pdf](http://www.macromedia.com/desdev/mx/studio/whitepapers/rich_internet_apps.pdf).
62. Macromedia. <http://www.macromedia.com/software/flash/>.
63. Macromedia. Macromedia ColdFusion MX.  
<http://www.macromedia.com/software/coldfusion/>.
64. Macromedia. Macromedia JRun 4. <http://www.macromedia.com/software/jrun/>.
65. Macromedia. Macromedia Showcase. <http://www.macromedia.com/showcase/>.
66. Macromedia. Rich Internet Applications Development Center.  
<http://www.macromedia.com/desdev/ria/>.
67. Macromedia. Rich Internet Applications Web Site.  
[http://www.macromedia.com/resources/business/rich\\_internet\\_apps/](http://www.macromedia.com/resources/business/rich_internet_apps/).
68. Manchester Airport. Arrivals/Departures Information.  
[http://www.manchesterairport.co.uk/content.nsf/Arrivals\\_Departures!ReadForm](http://www.manchesterairport.co.uk/content.nsf/Arrivals_Departures!ReadForm).
69. Marcus, A., and Chen, E. Designing the PDA of the Future. Interactions, January-February 2002, pp. 34-44.

70. Marcus, A., Ferrante, J. V., Kinnunen, T., Kuutti, K., and Sparre, E. Baby Faces: User-Interface Design for Small Displays. Proceedings of the 1998 SIGCHI Conference on Human Factors in Computing Systems, Los Angeles, 1998.
71. McAlester, D. and Capraro, M. *Skip Intro: Flash Usability and Interface Design*, New Riders, April 2002.
72. Menkhaus, G., and Pree, W. User Interface Tailoring for Multi-Platform Service Access. Proceedings of the 7<sup>th</sup> International Conference on Intelligent User Interfaces, San Francisco, 2002, pp. 208-209.
73. Microcell Solutions, Inc. <http://www.fido.ca/portal/home/homepage.jsp?lang=en>.
74. Microsoft Corporation. Customer Fact Sheet on Removal of Java from Windows. February 26, 2003. <http://www.microsoft.com/windowsxp/pro/evaluation/news/jre.asp>.
75. Microsoft. Microsoft .NET. <http://www.microsoft.com/net/>.
76. Morrison, M., Morrison, J., and Keys, A. Integrating Web Sites and Databases. Communications of the ACM, Volume 45, Number 9, September 2002, pp. 81-86.
77. Mozilla.org. XRE (XUL Runtime Environment). January 2003. <http://www.mozilla.org/projects/xul/xre.html>.
78. Myers, B., McDaniel, R. G., Miller, R. C., Ferreny, A. S., Faulring, A., Kyle, B. D., Mickish, A., Klimovitski, A., and Doane, P. The Amulet Environment: New Models for Effective User Interface Software Development. IEEE Transactions on Software Engineering, Volume 23, Issue 6, June 1997.
79. Nakajima, T. A Middleware Component Supporting Flexible User Interfaction for Networked Home Appliances. ACM SIGARCH Computer Architecture News, Volume 29, Issue 5, December 2001.
80. Neches, R., Brown, J. S., Sondheimer, N., Malone, T., and Williams, M. Intelligence in Interfaces. Proceedings of the SIGCHI/GI Conference on Human Factors in Computing Systems and Graphics Interface, Toronto, 1987.
81. Oeschger, I. XUL Widget Cheatseet. September 2000. [http://www.mozilla.org/docs/xul/xulnotes/xulnote\\_cheatsheet.html](http://www.mozilla.org/docs/xul/xulnotes/xulnote_cheatsheet.html).
82. Olsen, D. J. A Programming Language Basis for User Interface Management. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Wings for the Mind, 1989.
83. Oracle Corporation. <http://www.oracle.com/>.

84. Palm, Inc. <http://www.palm.com/us/>.
85. Phelps, T. A., and Wilensky, R. The Multivalent Browser: A Platform for New Ideas. Proceedings of the 2001 ACM Symposium on Document Engineering, Atlanta, 2001, pp. 58-67.
86. Pocket PC Magazine. Pocket PC Phone Edition & Windows Powered Smartphones "At a Glance": A Feature by Feature comparison. [http://www.pocketpcmag.com/bg2003/phones.asp?sort=CPU\\_Speed](http://www.pocketpcmag.com/bg2003/phones.asp?sort=CPU_Speed).
87. Puerta, A., and Szekely, P. Model-Based Interface Development. Proceedings of the 1994 SIGCHI Conference on Human Factors in Computing Systems, Boston, 1994.
88. Quaker Oats Company. Kid's Real Life Food Pyramid. [http://www.quakeroatmeal.com/PartnersNutrition/KFP\\_App.cfm](http://www.quakeroatmeal.com/PartnersNutrition/KFP_App.cfm).
89. Rees, Michael J. Evolving the Browser Towards a Atandard User Interface Architecture. Third Australian Conference on User Interfaces, Volume 7, Melbourne, 2002, pp. 1-7.
90. Refsnes Data. Introduction to SOAP. [http://www.w3schools.com/soap/soap\\_intro.asp](http://www.w3schools.com/soap/soap_intro.asp).
91. Refsnes Data. Introduction to WSDL. [http://www.w3schools.com/wSDL/wSDL\\_intro.asp](http://www.w3schools.com/wSDL/wSDL_intro.asp).
92. Refsnes Data. WAP/WML Tutorial. <http://www.w3schools.com/wap/>.
93. Research In Motion. <http://www.rim.com/>.
94. Rice, J., Farquhar, A., Piernot, P., and Gruber, T. Using the Web Instead of a Window System. Proceedings of the SIGCHI Conference on Human factors in Computing Systems, Vancovuer, 1996, pp. 103-110.
95. Robertson, G. G., and Mackinlay, J. D. The Document Lens. Proceedings of the 6<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology, Atlanta, 1993.
96. Rogers Communications, Inc. <http://www.shoprogers.com/store/wireless/overview.asp>.
97. Rose, G., Khoo, H., and Straub, D. Current Technological Impediments to Business-to-Consumer Electronic Commerce. Communications of the Association for Information Systems, Volume 1, Number 16, June 1999, Article 16.

98. Samulowitz, M., Michahelles, F., and Linnhoff-Popien, C. Adaptive Interaction for Enabling Pervasive Services. 2<sup>nd</sup> ACM International Workshop on Data Engineering for Wireless and Mobile Access, Santa Barbara, 2001, pp. 20-26.
99. Sarkar et al. Stretching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens. Proceedings of the 6<sup>th</sup> Annual ACM Symposium on User Interface Software and Technology, Atlanta, 1993, p. 81-91.
100. Schilit, B. N., Trevor, J., Hilbert, D. M., and Tzu, K. K. m-Links: An Infrastructure for Very Small Internet Devices. Proceedings of the 7<sup>th</sup> Annual International Conference on Mobile Computing and Networking, Rome, 2001, pp. 122-131.
101. Schuster, J. Defining User Interface/Application Interactions. April 2002. [http://www.harmonia.com/resources/papers/Defining\\_UI-App\\_Interactions\\_v02.pdf](http://www.harmonia.com/resources/papers/Defining_UI-App_Interactions_v02.pdf).
102. Shim, R. Removable Flash Cards Continue to Shrink. ZD Net Australia Reviews, 11 December 2002. <http://www.zdnet.com.au/reviews/computers/storage/story/0,2000023527,20270599,00.htm>.
103. Singh, G. and Green, M. Chisel: A System for Creating Highly Interactive Screen Layouts. Proceedings of the 2<sup>nd</sup> Annual ACM SIGGRAPH Symposium on User Interface Software and Technology, Williamsburg, 1989.
104. Spence, R. A Taxonomy of Graphical Presentation.
105. Steinman, D. The Dynamic Duo: Cross-Browser Dynamic HTML. <http://www.dansteinman.com/dynduo/>.
106. Su, N. M., Sakane, Y., Tsukamoto, M., and Nishio, S. Rajicon: Remote PC GUI Operations Via Constricted Mobile Interfaces. Proceedings of the 8<sup>th</sup> Annual International Conference on Mobile Computing and Networking, Atlanta, 2002.
107. Sun Microsystems. <http://java.sun.com/>.
108. Sun Microsystems. Java 2 Platform, Micro Edition. <http://java.sun.com/j2me/j2me-ds.pdf>.
109. Sun Microsystems. Microsoft Agrees to Settlement that Protects Future Integrity of the Java Platform. <http://java.sun.com/lawsuit/>.
110. Swishzone.come. <http://www.swishzone.com/index.php>.
111. Techtarget.com. Definition of "Intranet". [http://searchwebservices.techtarget.com/sDefinition/0,,sid26\\_gci212377,00.html](http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci212377,00.html).

112. Trevor, J., Hilbert, D. M., Schilit, B. N., and Tzu, K. K. From Desktop to Phonetop: A UI for Web Interaction on Very Small Devices. Proceedings of the 14<sup>th</sup> Annual ACM SIGGRAPH Symposium on User Interface Software and Technology, Orlando, 2001.
113. Turau, V. A Framework for Automatic Generation of Web-based Data Entry Applications Based on XML. Proceedings of the 2002 ACM Symposium on Applied Computing, Madrid, 2002, pp. 1121-1126.
114. Upsdell, C. Browser News. <http://www.upsdell.com/BrowserNews/browsers.htm>.
115. VoiceXML Forum. <http://www.voicexml.com/>.
116. Vuorimaa, P., Ropponen, T., von Knorring, N., and Honkala, M. A Java Based XML Browser for Consumer Devices. Proceedings of the 2002 ACM Symposium on Applied Computing, Madrid, 2002, pp. 1094-1099.
117. WebMethods, Inc. <http://www.webmethods.com/>.
118. Wiecha, C., and Boies, S. Generating User Interfaces: Principles and Use of Its Style Rules. Proceedings of the 3<sup>rd</sup> Annual ACM SIGGRAPH Symposium on User Interface Software and Technology, Snowbird, 1990.
119. Wiecha, C., Bennett, W., Boies, S., and Gould, J. Generating Highly Interactive User Interfaces. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Wings for the Mind, 1989.
120. WizeGuides.com. Interactive subway map of Boston. <http://www.wizeguides.com/wizeguides/>.
121. World Wide Web Consortium. A Little History of the World Wide Web. 1995. <http://www.w3.org/History.html>.
122. World Wide Web Consortium. About SVG: 2d Graphics in XML. 2004. <http://www.w3.org/Graphics/SVG/About.html>.
123. World Wide Web Consortium. SVG Implementations. 2004. <http://www.w3.org/Graphics/SVG/SVG-Implementations.htm8>.
124. World Wide Web Consortium. XForms: The Next Generation of XForms. 2004. <http://www.w3c.org/MarkUp/Forms/>.
125. XMethods, Inc. <http://www.xmethods.com/>.

126. Adobe Systems. Adobe Form Server: Easy Design, Delivery, and Processing of Electronic Forms. 2004.  
[http://www.adobe.com/products/server/formserver/pdfs/formserver\\_whitepaper.pdf](http://www.adobe.com/products/server/formserver/pdfs/formserver_whitepaper.pdf).

## APPENDIX 1

### *SUIML: Scalable User Interfaces Markup Language*

#### Notes

- SUIML is XML-based; all tags must be closed either with a closing tag as in `<tag attributes...> ... </tag>` or with a closing slash as in `<tag attributes... />`.
- Case is not relevant for tag and attribute names
- All tags have two required attributes that are common across all tags:
  - `name`: a unique name referring to the form object – this is later used if callbacks from Web service method invocations.
  - `label`: all form objects have a label which are used in different ways for different objects.
- Some tags are implemented in the current version of SUI while others are left for future implementation:
  - Implemented
    - `application`
    - `box`
    - `button`
    - `checkbox`
    - `image`



- item
- label
- radio
- radioGroup
- text
- select
- Left for future implementation
  - dialog
  - tabGroup
  - tabPanel
  - splitGroup
  - splitPanel
- Other tags are under consideration but have not been defined as part of the SUIML language:
  - grid
    - column
    - row
      - rowEntry
  - calendar
  - menu

## ***application***

This tag needs to be the first tag in a form definition if the form will invoke a back-end Web service.

### **Required Attributes**

<code>service</code>	The URL of the WSDL definition file for the Web service
----------------------	---

### **Status**

Left for future implementation.

## ***box***

A grouping of form elements which are laid out as a collective whole; they cannot be separated in the layout.

### **Required Attributes**

<code>name</code>	A unique name referring to the box
<code>label</code>	A label for the box; will be displayed as the title of the box

### **Optional Attributes**

<code>border</code>	Indicates if the box should have a border; default is false. Possible values are true or false.
---------------------	---

## Possible Parents

This tag can be contained in the following tags:

- box
- tabPanel
- splitPanel
- dialog

## Child Elements

This tag can contain the following tags:

- box
- tab
- image
- button
- radioGroup
- checkbox
- text
- label
- split
- dialog
- select

**Status**

Implemented.

***button***

A button.

**Required Attributes**

Name	A unique name referring to the button
Label	A label for the button; will be displayed as the text displayed in the button

**Optional Attributes**

clickMethod	Web service method to call when the button is clicked; name of the field will be passed as an argument to the method.
-------------	---

***checkbox***

A check box.

**Required Attributes**

name	A unique name referring to the check box
label	A label for the check box; will be displayed as the text for the check box

value	The value associated with the radio button which is sent as an argument to Web service methods
-------	--

### Optional Attributes

selectMethod	Web service method to call when the radio button is selected; name of the field will be passed as an argument to the method.
unselectMethod	Web service method to call when the radio button is unselected; name of the field will be passed as an argument to the method.

### Possible Parents

This tag can be contained in the following tags:

- box
- tabPanel
- splitPanel
- dialog

### Child Elements

This tag can contain the following tags: none.

### Status

Implemented.

### *dialog*

A dialog box.

### Required Attributes

<code>name</code>	A unique name referring to the dialog box
<code>label</code>	A label for the dialog box; will be displayed as the title of the dialog box

### Optional Attributes

<code>openMethod</code>	Web service method to call when the dialog box is opened; name of the field will be passed as an argument to the method.
<code>closeMethod</code>	Web service method to call when the dialog box is closed; name of the field will be passed as an argument to the method.

### Possible Parents

This tag can be contained in the following tags:

- `box`
- `tabpanel`
- `splitPanel`
- `dialog`

### Child Elements

This tag can contain the following tags:

- `box`
- `tab`

- image
- button
- radioGroup
- checkbox
- text
- label
- split
- dialog
- select

### Status

Left for future implementation.

### ***image***

An image (a static image field).

### Required Attributes

name	A unique name referring to the image field
label	A label for the field; will be displayed as the caption of the field

## Optional Attributes

scale	Percentage scaling of the image; default is 100%
-------	--

## Possible Parents

This tag can be contained in the following tags:

- box
- tabPanel
- splitPanel
- dialog

## Child Elements

This tag can contain the following tags: none.

## Status

Implemented.

## *item*

A item in a selection list.

## Required Attributes

name	A unique name referring to the item
label	A label for the item; will be displayed as the text for the item



value	The value associated with the item which is sent as an argument to Web service methods
-------	--

### Optional Attributes

selectMethod	Web service method to call when the item is selected; name of the field will be passed as an argument to the method.
unselectMethod	Web service method to call when the item is unselected; name of the field will be passed as an argument to the method.

### Possible Parents

This tag can be contained in the following tags:

- select

### Child Elements

This tag can contain the following tags: none.

### Status

Implemented.

### ***label***

A label (a static text field).

### Required Attributes

name	A unique name referring to the label field
------	--

<code>name</code>	A unique name referring to the label field
<code>label</code>	A label for the field; will be displayed as the text of the field

### Optional Attributes

<code>size</code>	Number of characters per line in which should be viewable; the default is to allow the rendering engine to determine this value.
<code>lines</code>	The number of lines to display; the default is a single-line text field.

### Possible Parents

This tag can be contained in the following tags:

- `box`
- `tabPanel`
- `splitPanel`
- `dialog`

### Child Elements

This tag can contain the following tags: none.

### Status

Implemented.

## ***radio***

A radio button.

### **Required Attributes**

<code>name</code>	A unique name referring to the radio button
<code>label</code>	A label for the radio button; will be displayed as the text for the radio button
<code>value</code>	The value associated with the radio button which is sent as an argument to Web service methods

### **Optional Attributes**

<code>selectMethod</code>	Web service method to call when the radio button is selected; name of the field will be passed as an argument to the method.
<code>unselectMethod</code>	Web service method to call when the radio button is unselected; name of the field will be passed as an argument to the method.

### **Possible Parents**

This tag can be contained in the following tags:

- `radioGroup`

### **Child Elements**

This tag can contain the following tags: none.

**Status**

Implemented.

***radioGroup***

A group of related radio buttons.

**Required Attributes**

name	A unique name referring to the group of radio buttons
label	A label for the group of radio buttons; will be displayed as an overall prompt for the group

**Possible Parents**

This tag can be contained in the following tags:

- box
- tabPanel
- splitPanel
- dialog

**Child Elements**

This tag can contain the following tags:

- radio

**Status**

Implemented.

## ***tabGroup***

A group of related tab panels.

### **Required Attributes**

<code>name</code>	A unique name referring to the group of tab panels
<code>label</code>	A label for the group of tab panels; will be displayed as an overall title for the group

### **Possible Parents**

This tag can be contained in the following tags:

- `box`
- `tabPanel`
- `splitPanel`
- `dialog`

### **Child Elements**

This tag can contain the following tags:

- `tabPanel`

### **Status**

Left for future implementation.

## ***tabPanel***

A tab panel.

### Required Attributes

name	A unique name referring to the tab panel
label	A label for the tab pabel; will be displayed as the text in the tab of the tab pane

### Optional Attributes

selectMethod	Web service method to call when the tab panel is selected; name of the field will be passed as an argument to the method.
unselectMethod	Web service method to call when the tab panel is unselected; name of the field will be passed as an argument to the method.

### Possible Parents

This tag can be contained in the following tags:

- tabGroup

### Child Elements

This tag can contain the following tags:

- box
- tab
- image
- button
- radioGroup

- checkbox
- text
- label
- split
- dialog
- select

**Status**

Left for future implementation.

***text***

A text field for entering text.

**Required Attributes**

name	A unique name referring to the text field
label	A label for the text field; will be displayed as a prompt for the text field

**Optional Attributes**

size	Number of characters per line in which should be viewable; the default is to allow the rendering engine to determine this value.
------	--

<code>lines</code>	The number of lines to display; the default is a single-line text field.
<code>changeMethod</code>	Web service method to call when the form field value changes; name of the field will be passed as an argument to the method.

### Possible Parents

This tag can be contained in the following tags:

- `box`
- `tabPanel`
- `splitPanel`
- `dialog`

### Child Elements

This tag can contain the following tags: none.

### Status

Implemented.

### ***select***

A selection list.

### Required Attributes

<code>name</code>	A unique name referring to the selection list
-------------------	---



label	A label for the list; will be displayed as the prompt for the list.
-------	---

### Possible Parents

This tag can be contained in the following tags:

- box
- tabPanel
- splitPanel
- dialog

### Child Elements

This tag can contain the following tags:

- item

### Status

Implemented.

### ***splitGroup***

A group of related split panels.

### Required Attributes

name	A unique name referring to the group of split panels
label	A label for the group of split panels; will be displayed as an overall title for the group

## Possible Parents

This tag can be contained in the following tags:

- `box`
- `tabPanel`
- `splitPanel`
- `dialog`

## Child Elements

This tag can contain the following tags:

- `splitPanel`

## Status

Left for future implementation.

## ***splitPanel***

A split panel.

## Required Attributes

<code>name</code>	A unique name referring to the split panel
<code>label</code>	A label for the split panel; will be displayed as the header in the split panel

## Optional Attributes

<code>selectMethod</code>	Web service method to call when the split panel is selected; name of the field will be passed as an argument to the method.
<code>unselectMethod</code>	Web service method to call when the split panel is unselected; name of the field will be passed as an argument to the method.

## Possible Parents

This tag can be contained in the following tags:

- `splitGroup`

## Child Elements

This tag can contain the following tags:

- `box`
- `tab`
- `image`
- `button`
- `radioGroup`
- `checkbox`
- `text`
- `label`
- `split`
- `dialog`

- select

**Status**

Left for future implementation.