USER INTERFACES FOR MULTIMODAL SYSTEMS

BY

SUMANTH LINGAM

BACHELOR OF TECHNOLOGY IN CIVIL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, 1999

SUBMITTED TO THE DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Engineering
IN CIVIL AND ENVIRONMENTAL ENGINEERING

AT THE
Massachusetts Institute of Technology
JUNE 2001

SIGNATURE OF AUTHOR: _____

SUMANTH LINGAM
DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING
MAY 18, 2001

CERTIFIED BY: _____

KEVIN AMARATUNGA
ASSISTANT PROFESSOR, CIVIL AND ENVIRONMENTAL ENGINEERING
THESIS SUPERVISOR

CERTIFIED BY: _____

ORAL BUYUKOZTURK
CHAIRMAN, DEPARTMENTAL COMMITTEE ON GRADUATE STUDIES

USER INTERFACES FOR MULTIMODAL SYSTEMS

By
SUMANTH LINGAM

Submitted to the Department of Civil and Environmental Engineering Department on May 18, 2001,
in partial fulfillment of the requirements for the degrees of Master Of Engineering in Civil and Environmental Engineering

Abstract

As computer systems become more powerful and complex, efforts to make computer interfaces more simple and natural become increasingly important. Natural interfaces should be designed to facilitate communication in ways people are already accustomed to using. Such interfaces allow users to concentrate on the tasks they are trying to accomplish, not worry about what they must do to control the interface. Multimodal systems process combined natural input modes- such as speech, pen, touch, manual gestures, gaze, and head and body movements- in a coordinated manner with multimedia system output.

The initiative at W3C is to make the development of interfaces simple and easy to distribute applications across the Internet in an XML development environment. The languages so far such as HTML designed at W3C are for a particular platform and are not portable to other platforms. User Interface Markup Language (UIML) has been designed to develop cross-platform interfaces. It will be shown in this thesis that UIML can be used not only to develop multi-platform interfaces but also for creating multimodal interfaces. A survey of existing multimodal applications is performed and an efficient and easy-to-develop methodology is proposed. Later it will be also shown that the methodology proposed satisfies a major set of requirements laid down by W3C for multimodal dialogs.

Thesis Supervisor: Kevin Amaratunga
Title: Assistant Professor

Acknowledgments

I would like to thank my thesis advisor Prof. Kevin Amaratunga for his constant support and guidance while pursuing this thesis.

I would like to thank Harmonia, Inc. for providing me the required software at no price.

I would like to specially thank Jim Nelson, a fellow Master of Engineering student and a good friend. Also I would like to thank all my colleagues in Information Technology program.

I would like to thank my parents and my brothers for what I am today.

Table of Contents

1 Introduction 6

1.1 Overview 7

1.2 Preamble 7

1.2.1 The Problem statement 7

1.2.2 Multimodal Interfaces 8

1.3 Thesis Overview 9

2 User Interface Markup Language 11

2.1 Scenario 11

2.2 Ideal Solution 11

2.3 Motivation behind the design of UIML 13

2.4 Language Syntax 13

2.5 Features and Advantages of UIML 15

2.5.1 Primary Features 15

2.5.2 Advantages 15

2.6 UIML tools and Implementations 21

2.6.1 Renderer 21

2.6.2 Integrated Development Environment 21

2.6.3 Interface Server 22

2.7 Alternatives to UIML 23

2.7.1 Transcoding 23

2.7.2 XForms 24

2.8 Perspective on UIML 24

2.9 Complete example of UIML for target platforms 25

2.9.1 User interface 25

2.9.2 Virtual UI tree 27

2.9.3 UI Metaphor for HTML 28

2.9.4 UI Metaphor for WML 29

2.9.5 Target Code and Interface Generated for HTML 30

2.9.6 Target Code and Interface generated for WML 32
3 Survey of Existing Methodologies and Applications 34
3.1 Overview 34
3.2 Various Architectures for Multimodal Systems 34
3.2.1 Early Fusion or Feature Fusion 34
3.2.2 Late Fusion or Semantic fusion 34
3.2.3 Multi-agent Architectures 35
3.3 MMI System – A Case Study 37
3.3.1 Introduction and Purpose 37
3.3.2 System Architecture 37
3.3.3 Methodology 39
3.3.4 Analysis 40
4 Methodology 42
4.1 Overview 42
4.2 Prolog 42
4.3 Proposed Architecture 43
4.4 Proposed Methodology 45
4.5 Example 45
4.6 W3C's Requirements for Authoring Multimodal Dialogs 51
4.6.1 General Requirements 51
4.6.2 Input Modality requirements 58
4.6.3 Output Media Requirements 61
4.6.4 Architecture, Integration and Synchronization points 62
5 Conclusions 66
6 References 68

List of Figures

Figure 1 UIML Model vs. MVC 17
*Figure 2 M*N problem* 18
Figure 3 Connection to External Logic 20
Figure 4 Deployment and Rendering Architecture 22
Figure 5 Transcoding Process 23
Figure 6 XForms vs. UIML 24
Figure 7 Perspective on UIML 25
Figure 8 UI Tree 27
Figure 9 HTML Interface 31
Figure 10 WML Interface 33
Figure 11 Information Flow in a Multimodal System 35
Figure 12 Open Agent Architecture for Multimodal Integration 36
Figure 13 Architecture of MMI System 38
Figure 14 Architecture for Rendering Target Language Interfaces 44
Figure 15 Architecture for Rendering Multimodal Interfaces 44
Figure 16 UI tree Representation of Initial Interface 48
Figure 17 State 1 of the Dialog 50
Figure 18 State 2 of the Dialog 50
Figure 19 State 3 of the Dialog 51

1 Introduction

1.1 Overview

This thesis examines the various methodologies and multimodal applications developed over the past few years and suggest an efficient architecture and methodology to develop multimodal interfaces using User Interface Markup language in the World Wide Web environment.

1.2 Preamble

1.2.1 The Problem statement

The world is changing. People no longer want to be chained to their desktop computers to access information. Instead, they want to freely move through the day using Internet-connected appliances ranging from phones to palm PCs. With the advent of integrated data and voice carrier technologies like VoIP, VoATM and 3G Networks, there has been a proliferation of devices with distinguishable input and output capabilities in various modalities. This is leading us to a world of "ubiquitous networking" with data and voice access to every one, everywhere. The appropriate use of the modalities has become crucial for natural means of communication with the devices because different devices have different capabilities. A cell phone has voice capabilities equipped with small display, a PDA has only a display and a plain old telephone has only voice capabilities. Making use of most of the capabilities of a platform but at the same time not having to write interfaces customized for each device or platform is a challenging problem faced today. Consider a cell phone with both voice and display capabilities. A characteristic of speech input that it is very effective in its ability to bypass multiple layers of menus. At the same time it is frustrating for the user to repeat him self in case of wrong interpretation or recognition, while input through keypad could have been more effective in this situation. Also, the inherent serial nature of speech output makes it a laborious way of presenting information that could be quickly browsed on a display. Hence it is desired to develop user interfaces for systems or devices that have multimodal capabilities. Though in the past many multimodal applications were developed, it was difficult to extend such applications to other platforms mainly because they use device specific API's. With the introduction of Markup languages, effort to develop user

interfaces has been tremendously reduced. But again these languages were designed to suite a particular device or platform.

The problem pursued in this thesis is to develop an efficient methodology to develop user interfaces for multimodal systems using easy to use XML user interface languages that are truly platform independent to support all the modes available.

1.2.2 Multimodal Interfaces

As applications generally have become more complex, a single modality does not permit the user to interact effectively across all tasks and environments (Larson, Ferro, & Oviatt, 1999). A multimodal interface offers the user freedom to use a combination of modalities, or to switch to a better-suited modality, depending on the specifics of the task or environment. Since individual input modalities are well suited in some situations, and less ideal or even inappropriate in others, modality choice is an important design issue in a multimodal system.

Among other things, speech input offers speed, high-bandwidth information, and relative ease of use. It also permits the user's hands and eyes to be busy with a task, which is particularly valuable when users are in motion or in natural field settings. Users tend to prefer speech for functions like describing objects and events, sets and subsets of objects, out-of-view objects, conjoined information, past and future temporal states, as well as for issuing commands for actions or iterative actions (Cohen & Oviatt, 1995; Oviatt & Cohen, 1991). During multimodal keypad/pen/voice interaction, users tend to prefer entering descriptive information via speech, although their preference for pen/keypad input increases for digits, symbols, and graphic content (Oviatt & Olsen, 1994; Oviatt, 1997; Suhm, 1998).

While also supporting portability, pen/keypad input has a somewhat different and multi-functional range of capabilities. Although the pen/keypad can be used to write words that are analogous to speech, it also can be used to convey symbols and signs (e.g., digits, abbreviations), gestures, simple graphics and artwork, and to render signatures. In addition, it can be used to point, to select visible objects like the mouse does in a direct manipulation interface, and as a means of microphone engagement for speech input. From a usage standpoint, pen input provides a more private and socially-acceptable form

of input in public settings, and a viable alternative to speech under circumstances of extreme noise (Holzman, 1999; Gong, 1995).

In summary, multimodal interfaces should:

- Permit flexible use of input modes, including alternation and integrated use
- Support improved efficiency, especially when manipulating graphical information
- Can support less disfluent, shorter, and more linguistically-simplified constructions than a speech-only interface, which results in more robust natural language processing
- Support greater precision of spatial information than a speech-only interface, because pen/keypad input conveys rich and precise graphical information
- Satisfy higher levels of user preference
- Support enhanced error avoidance and ease of error resolution
- Accommodate a wider range of users, tasks, and environmental situations
- Are adaptable during the continuously changing environmental conditions of mobile use
- Accommodate individual differences, such as permanent or temporary handicaps
- Prevent overuse of any individual mode during extended computer usage

1.3 Thesis Overview

The thesis starts with the specification of the problem under consideration in section 1.2.1. Then an overview of general features of different modes is presented in section 1.2.2. This section lays down a few requirements for multimodal systems.

In chapter 2, a meta-interface language called User Interface Markup Language is studied in detail. Different reasons for selecting this language for multi-modal interface development are listed. Then a complete example of a user interface developed in UIML is presented to show its platform independent nature.

In chapter 3, several existing methodologies and architectures are described along with a case study of a multimodal system called MMI. In chapter 4, a methodology and architecture is proposed for developing multimodal user interfaces using User Interface

markup language. Also an analysis is performed to see if UIML satisfies the requirements specified by W3C for Multimodal Dialog Markup Language.

2 User Interface Markup Language

2.1 Scenario

A typical scenario in today's software industry is presented below, in order to better understand the philosophy behind the development of a language – User Interface Markup Language

Suppose the employer asked the development team to design multi-device portal for new e-commerce application, constraint imposed on project completion time and features must include rich user experience, content from XML via directories/databases and support for international customers.

A possible sequence of problems that the development team faces is

- Week1: No one on the team knows Palm API
- Week2: CEO tells the press that the company will also support voice
- Week3: The lead programmer quits
- Week4: Marketing says customers want easy customization of UI's across platforms
- Week5: and soon…

In addition to the above, other aspects the team has to cope up with are

- Training people in four languages, then realizing that new versions of the languages are released.
- Maintaining four source code bases
- Making UI's consistent across devices
- What if Palm loses in the marketplace?

2.2 Ideal Solution

The goal of all User interface languages (Java Swing/AWT, WML, VoiceXML, HTML) is to describe

- UI 'appearance'
- User interaction with UI

- Connection to
    - Data sources
    - Application logic

Though most of the existing UI languages account for the above features, they are not described but mixed with each other in syntax and style. There isn't clear separation of content, presentation and logic of a UI. For this reason none of the above languages are cross-platform and have been designed to suite one particular platform focusing on the underlying capabilities of the platform. In past, attempts have been made to design cross-platform languages but none of them were truly platform independent. A good example is Java. Java was originally designed to support various platforms but in realization, it is still widely used only on desktop computers for operating systems such as Linux, Windows etc. This partly cross-platform nature of Java affected its run-time performance adversely. Also its adoption for handheld and wireless devices has been restricted due to the bandwidth and computational resource constraints. So lightweight languages like HDML, WML were adopted instead to build applications for mobile devices. The necessity to have languages suited for target platforms but at the same time not having to develop UI applications in each language was the motivation to create a meta-language that is truly cross-platform and renders complete functionality of the target platforms. The result is User Interface Markup Language (UIML).

An Ideal UI Language is the one, which is

- Usable by non-programmers
- Allows reuse of good UI designs and patterns
- Extensible for future devices
- Can populate UI from XML via XSL, without scripting.
- Promote good UI design

All the above features are embodied in UIML. It is designed to be a UI language and not programming language and hence usable by non-programmers. UIML uses a template mechanism to store and reuse good UI designs and patterns. Since it is cross-platform language and driven only by the vocabulary of the target device, it is extensible for future

devices. A UI designed in UIML can populate content from XML files as will be seen in section 2.4.

2.3 Motivation behind the design of UIML

One might argue that an existing language could be augmented or modified to generate user interfaces for an arbitrary device; why design a new language?

One answer is that existing languages were designed with inherent assumptions about the type of user interfaces and devices for which they would be used. For example, HTML started as a language for describing documents (with tags for headings, paragraphs, and so on), and was augmented to describe forms. As another example, JavaScript events correspond to a PC with a GUI, mouse, and keyboard. In theory, it is possible to modify languages to handle any type of device, but this produces a stress on the language design. Witness the complexity added to HTML from version 2 to 4. Imagine the complexity if a future version of DHTML supported any device type.

A language like Java contains fewer assumptions and would be more feasible to use as a universal, device-independent language. But this would require Java to run on all devices (which may never occur), and device-specific code (e.g., for layout) would be needed. Moreover, Java is for experts and novice programmers must invest valuable time to master it.

Based on these arguments, it would be more natural to create a language that derives its syntax from XML. People are already familiar with XML due to the success of HTML. XML permits new tags to be defined, which is appealing in designing a language that must work with yet-to-be-invented devices.

2.4 Language Syntax

A typical UIML2 document is composed of these two parts:

A prolog identifying the XML language version and encoding and the location of the UIML2 document type definition (DTD):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

The root element in the document, which is the *uiml* tag:

<uiml xmlns='http://uiml.org/dtds/UIML2_0f.dtd'> ... </uiml>

The *uiml* element contains four child elements:

1. An optional header element giving metadata about the document:
<head> ... </head>

2. An optional user interface description, which describes the parts comprising the interface, and their structure, content, style, and behavior:
<interface> ... </interface>

3. An optional element that describes the mapping from each property and event name used elsewhere in the UIML2 document to a presentation toolkit and to the application logic:
<peers> ... </peers>

4. An optional element that allows reuse of fragments of UIML2:
<template> ... </template>

The Skeleton structure of UIML is

```
<uiml>
      <interface>
            <structure> .....</structure>
            <style>.....</style>
            <behavior>.....</behavior>
      </interface>
```

```
    <peers> .....</peers>
</uiml>
```

2.5 Features and Advantages of UIML

2.5.1 Primary Features

- Provides a canonical format for describing interfaces that map to multiple devices.
- Generates one description of a user interface connection to application logic independent of target device.
- Maps the interface description to a particular device/platform.
- Describes separately the content, structure, style, and behavior aspects of the interface.
- Describes behavior in a device-independent manner.
- Gives the same power as with the native toolkit.
- Connects one user interface description to multiple application logic.
- Connects multiple user interface descriptions to one application logic.

2.5.2 Advantages

2.5.2.1 UIML describes UI for a family of platforms

UIML describes UI for a family of platforms. A user interface in the future may be delivered on dozens of different devices. Some might have large screens and keyboards. Others might have small or no screens and only an alphanumeric keypad, or perhaps just a touch screen or voice input. It would be unreasonable for a user interface language to require different user interface descriptions for each device. Instead, the interface descriptions should be organized into a tree or other structure, with interfaces common to multiple devices factored out into "families." For example, cell phones; PDA's and other handheld devices have small displays. So the structure and layout of the UI on these devices are similar and easily captured in UIML.

2.5.2.2 UIML provides canonical representation of UI's

There are so many syntaxes for UI languages today. The most transparent and easy to use syntax is canonical and hence adopted into UIML.

2.5.2.3 UIML assists in UI design process

A developer starts the design process with certain aspects in mind such as what tasks the end user would do and what information and interactions are required. UIML can express the result of the design process. It captures the author's intent in a natural way. The tools should combine the design process with UIML.

2.5.2.4 UIML is Expressive

It permits creation of virtually any UI that could be created by handwriting any of the languages like Java, C++ (MFC, PalmOS API etc.), Visual basic, VoiceXML, HTML, WML etc. Therefore UIML is a substitute for any of these languages.

2.5.2.5 UIML model vs. MVC

Figure1 shows how UIML model is 6-way separated compared to the highly popular 3-way separated Model View Controller Architecture, which gives the developer the capability to develop and store highly customized interfaces based on user preferences or device's profile.

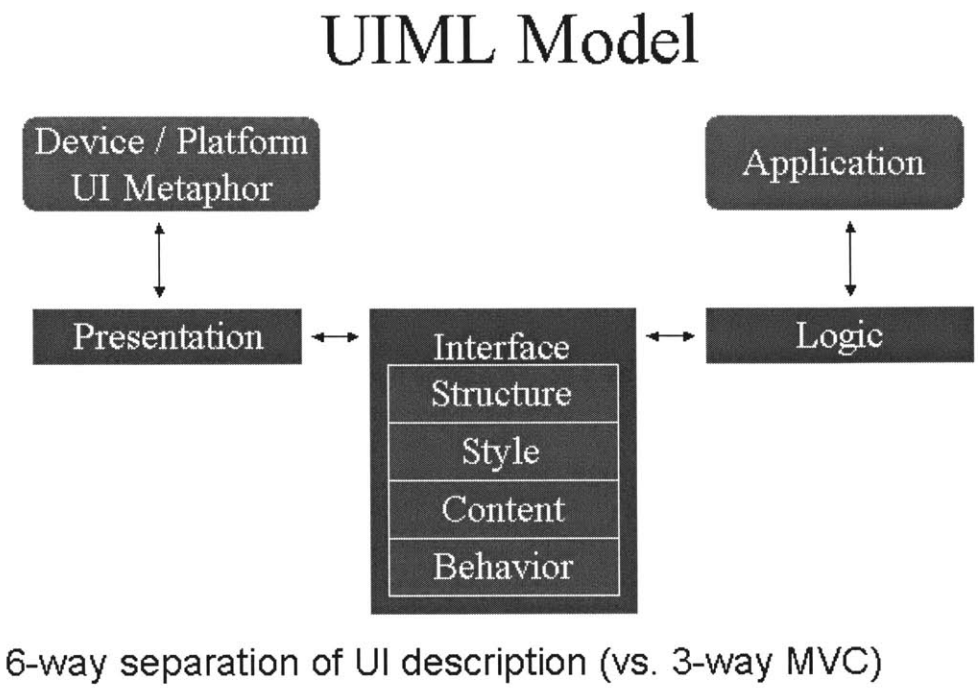


Figure 1 UIML Model vs. MVC

2.5.2.6 M*N Problem

UIML solves the classical M*N problem. From the above model we see that in UIML, a UI is separated into Structure, Style, Content and Behavior. Consider M interfaces to be written for each of N devices. With the existing languages, a developer will have to write M*N interfaces. With UIML, the developer writes M+N interfaces. This is possible because each of the M interfaces is written with a consistent developer's vocabulary and then his vocabulary is mapped to N presentation files for N devices.

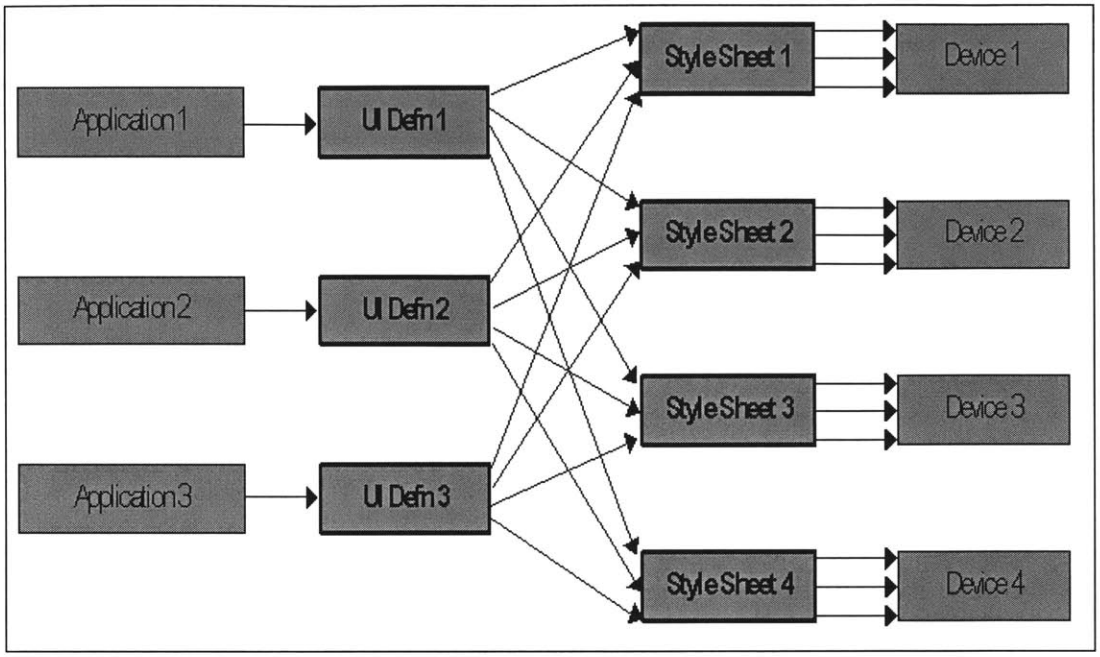


Figure 2 M*N problem

2.5.2.7 UIML assists in rapid Prototyping

UIML assists in Rapid Prototyping, an important concept in the software development process. User interface design teams often need to implement prototype user interfaces quickly to gain feedback from customers or end-users. A design methodology of iterative enhancement may be used, which requires interface changes to be made quickly and easily. Or a design team may use a scenario approach, in which an interface containing only sufficient functionality to support a scenario is created. UIML allows radical changes with minor modifications.

2.5.2.8 UIML allows Internationalization

UIML allows creation of Internationalized software because it separates content from appearance. A UIML2 document may specify multiple content elements in external

entities with different encodings. The entities are then included in the main document using the XML inclusion mechanism or the UIML2 source mechanism. For example,

```
<uiml>
   <interface>
     <content name="English" source="EnglishContent.uiml"/>
     <content name="Greek" source="GreekContent.uiml"/>
   </interface>
</uiml>
```

2.5.2.9 UIML allows custom vocabulary

Multiple target platform interfaces can be generated from one UIML file by using *Generic Vocabulary*. For example GButton widget name can be used for JButton in Java Swing, Command Button in PalmOS, <input type="Button"> in WML and <input type="Button"> in HTML.

2.5.2.10 UIML describes connection to Logic

With UIML, the developer can directly connect to the external data sources or application servers with out having to use CGI or Servlet programs. This is possible using the <call> statement in UIML with internal and external proxies, which take responsibility of handling the <call> statement on behalf of the User Interface.

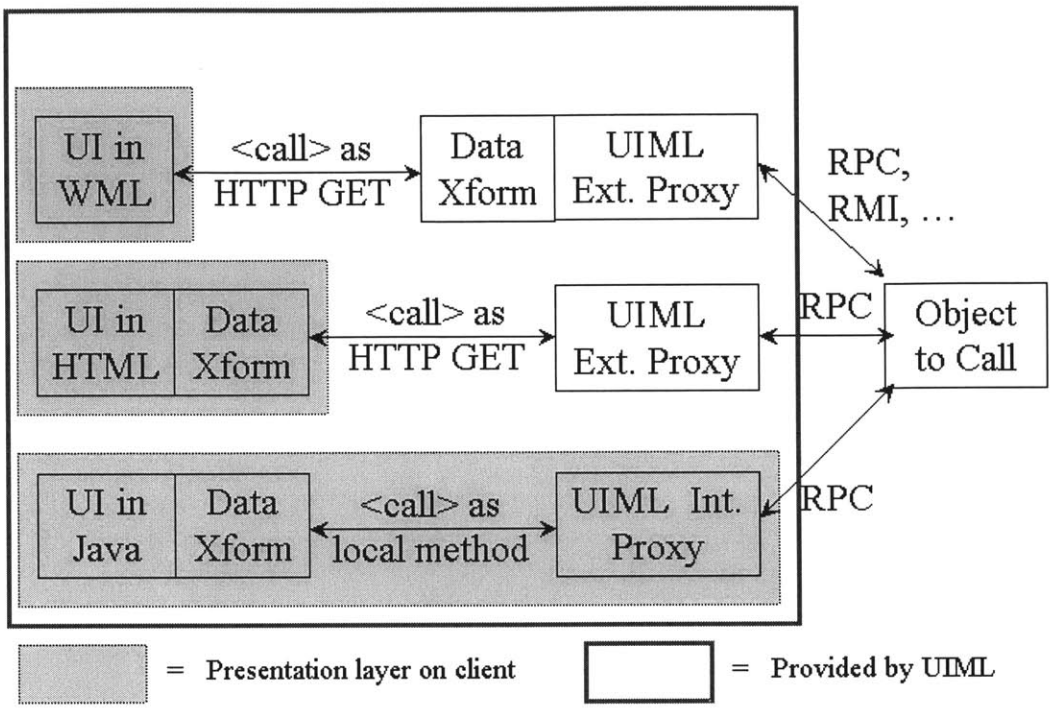


Figure 3 Connection to External Logic

2.5.2.11 UIML allows dynamic content

User interfaces can populate content statically using models and dynamically using <call> statements to get content from external data sources. An Example of populating dynamic content is given below.

```
<content>
   <constant name="table_1">
     <call name="myExample.getTable" transform-
name="formatTable" transform-mime="text/xsl">
        <param>table_1</param>
     </call>
   </constant>
   <constant name="formatTable">
     <xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
       <xsl:template match="/">
        ...
```

```
        </xsl:template>
      </xsl:stylesheet>
    </constant>
  </content>
```

2.5.2.12 UIML allows efficient download of interfaces over the Internet

UIML allows efficient download of user interfaces over networks to Web browsers. There are two ways to deliver interfaces to Web browsers: deliver code (e.g., Active-X, Java) or deliver HTML. Delivering code allows an arbitrarily complex interface to be rendered. However, code files are hundreds of kilobytes or larger and slow to download. Also, code is often not cached by browsers and proxy servers, thus wasting network bandwidth every time the interface is started. On the other hand, HTML files are typically small (tens of kilobytes) and cacheable, allowing relatively fast download, but HTML cannot generate as rich an interface as code. UIML achieves the flexibility of downloading code, but requires time and network bandwidth comparable to that required by HTML.

2.6 UIML tools and Implementations

2.6.1 Renderer

The Renderer is either a translator tool to translate a *single* meta-language user interface description into *one of several* platform-specific source codes or interpret UIML and render an interface at run-time. The translator tool can translate UIML to Java, WML, VoiceXML, HTML etc. The meta-translator tool could reside on the server or the client device. In a situation where the translator resides on the server, the translation of UIML file to target language file is done on the server and the translated UI file is transferred to the client. In a case where the translator resides on the client device, the translation is done on the client.

2.6.2 Integrated Development Environment

The IDE is a graphical software tool, which allows a developer to build user interfaces with drag and drop abilities. Firstly the developer is presented with a set of generic UI widgets with which, he/she can develop one interface. The interface is stored in one

UIML file and the mapping of the generic widgets to platform specific widgets is stored in pre-specified <peers> UIML files. Then the developer can preview the developed interface in various browsers. If the developer is unsatisfied, then he could shift to platform specific mode and modify a part of the interface into platform specific UIML code. These are stored as patch UIML files that can be integrated in to the main UIML file during translation. Also various intelligent agent modules will be assisting the developer during the process of interface development.

2.6.3 Interface Server

The UIML interface(s) developed are deployed on the Interface Server. When a device contacts the Interface Server, the device profile is read and the UIML interface is customized to best render the interface on the device. After customization, the interface server translates the UIML file into a platform specific User Interface by invoking the appropriate translator. The architecture of the rendering process is as shown below.

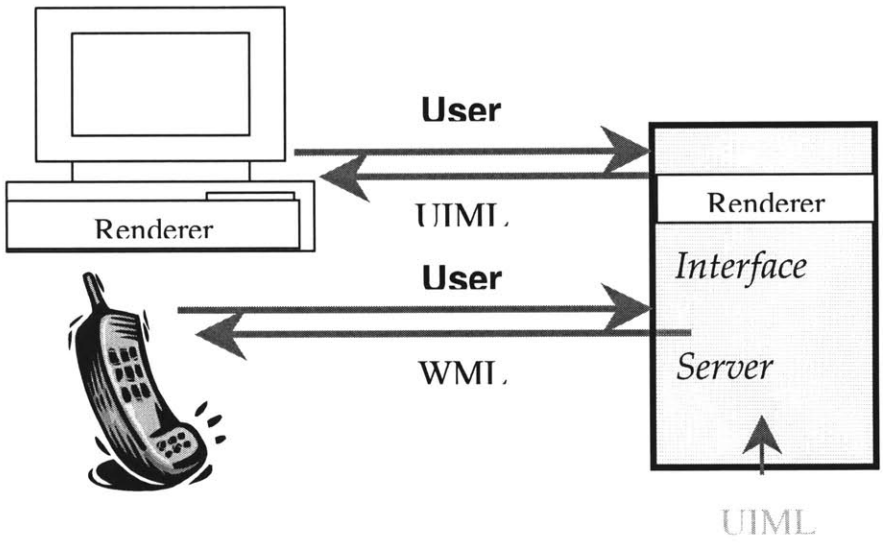


Figure 4 Deployment and Rendering Architecture

The Interface server also adapts a User Interface based on the user's spoken language, his/her role, user's organization, user's habits etc.

2.7 Alternatives to UIML

In the recent years several research and corporate organizations have tried to achieve similar goals of UIML. Most important derivatives of them are Transcoding proxies and XForms standard.

2.7.1 Transcoding

Today the solution used by the industries for web content adaptation for multiple devices is 'Transcoding', that is annotation based on Resource Description Framework (RDF). The figure below explains the procedure involved in Transcoding.

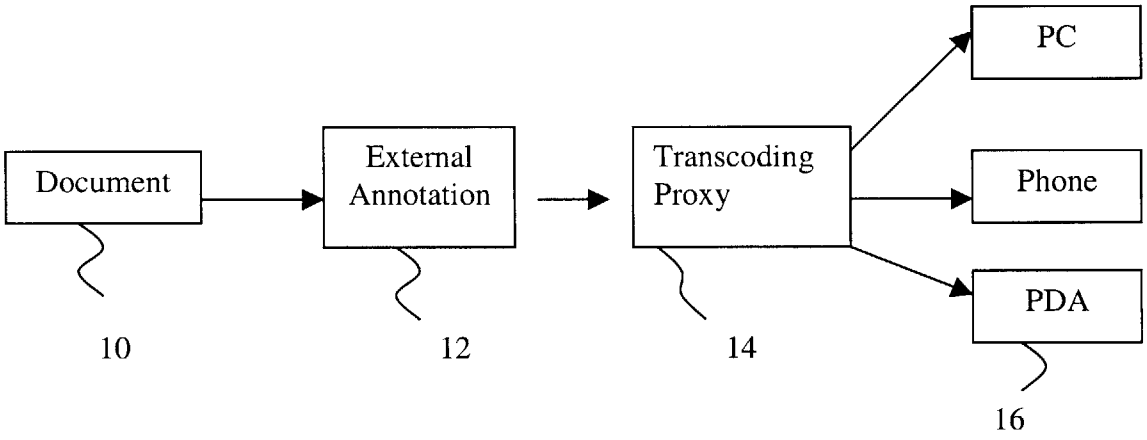


Figure 5 Transcoding Process

The original document 10 is either annotated externally 12 or with in the authoring tool during the creation of the document. Upon receiving a request from a personal device 16, a proxy server 14 may adapt the document on the basis of associated annotations. The rendered document is then downloaded to a client device with a small display 16. Though Transcoding provides a way to adapt content for display for mutiple devices, it does not describe an interface in a generic sense and must guess semantic information that's present in UIML. This means that through transcoding tools only the content can be adapted and not the structure, behavior and style of the interface. Also Transcoding is suitable for Legacy content and UIML for describing new UI's.

2.7.2 XForms

After UIML became public through UIML.org, there has been an initiative in W3C to have a language that is similar to UIML. The result is XForms. But there are quite a lot of differences between UIML and XForms as summarized in the table below.

XForms Versus UIML

XForms (2/16/01 draft)	UIML
Web-oriented UIs	Any UI
Content mixed into UI structure	Separates content
Sophisticated data model	Limited data models; could adopt Xform models
No event handling	Event handling included
UI only connects to form processor	UI connects to anything
Limited extensibility	Total extensibility

Figure 6 XForms vs. UIML

2.8 Perspective on UIML

The situation today is like the PC industry 20 years ago with many devices, many API's. Eventually Operating Systems shielded developers from device API's. Similarly, UIML provides a shield for devices.

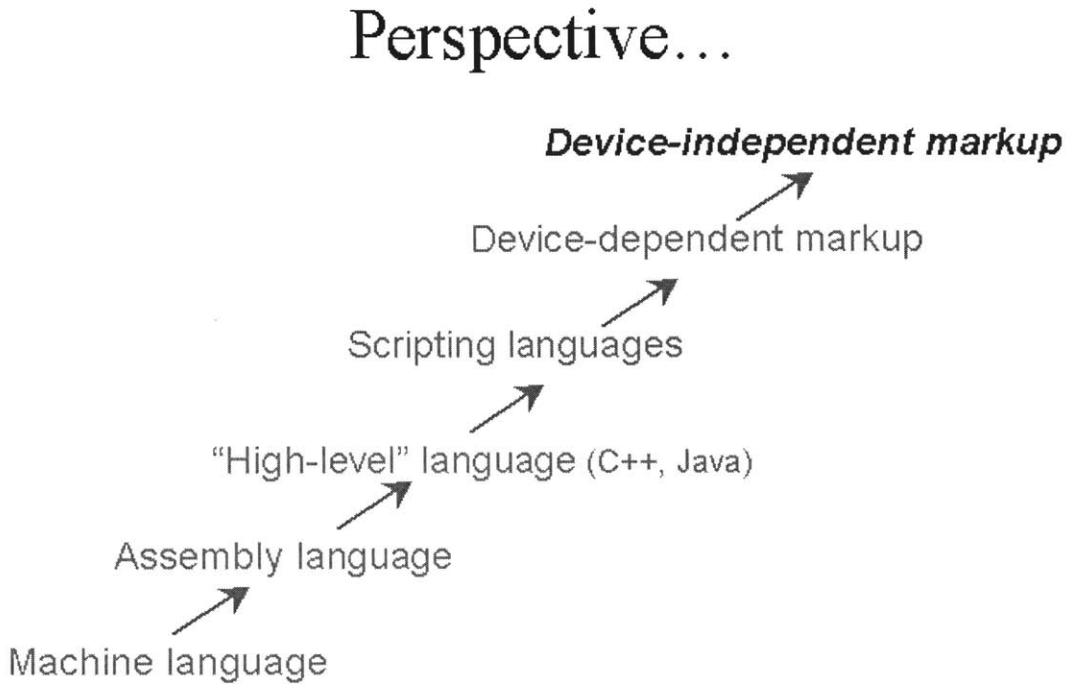


Figure 7 Perspective on UIML

2.9 Complete example of UIML for target platforms

In this section the usefulness of UIML is illustrated through a complete example. First the user interface is written in UIML. Then the mapping of the *generic UI widgets* is mapped to the platform specific *UI widgets.* The target languages considered for the example are HTML, which is primarily used to build interfaces to view web content on desktop platforms and WML, which is primarily used to build interfaces to view web content for mobile or handheld devices with small display size. This example is quite substantial to prove that UIML is the ideal language to build cross-platform user interfaces.

2.9.1 User interface

The user interface shown below is a front end to a web application that accepts the user's first, last, middle names and password for authentication.

```
<uiml>
 <interface>
   <structure>
   <part name="inputC" class="Container">
    <part name="inputB" class="Body">
     <part name="inputF" class="Form">
       <part name="titleP" class = "Paragraph"/>
       <part name="secondP" class="Paragraph">
         <part name="fname" class="Text"/>
         <part name ="i1" class="Input"/>
         </part>
         <part name="thirdP" class="Paragraph">
           <part name="mname" class="Text"/>
           <part name ="i2" class="Input"/>
         </part>
         <part name="fourthP" class="Paragraph">
           <part name="lname" class="Text"/>
           <part name ="i3" class="Input"/>
         </part>
         <part name="fifthP" class="Paragraph">
           <part name="passwd" class="Text"/>
           <part name ="i4" class="Input"/>
         </part>
        </part>
       </part>
     </part>
   </structure>
 <style>
   <property part-name="titleP" name="content">Log into site:</property>
   <property part-name="fname" name="content">First name:</property>
   <property part-name="i1" name = "value">Donald</property>
   <property part-name="mname" name="content">Middle initial:</property>
   <property part-name="i2" name = "value">D</property>
   <property part-name="lname" name="content">Last Name:</property>
   <property part-name="i3" name = "value">Duck</property>
   <property part-name="passwd" name="content">Password:</property>
   <property part-name="i4" name = "type">password</property>
   <property part-name="i4" name = "value">Mickey</property>
 </style>
 <behavior>
   <rule>
     <condition>
       <event part-name="inputF" class="accept"/>
     </condition>
     <action>
       <call name="form.submit">
        <param name="url">http://somewhere.com</param>
        <param name="method">get</param>
       </call>
     </action>
   </rule>
 </behavior>
</interface>
</uiml>
```

2.9.2 Virtual UI tree

Below is the virtual UI tree with nodes representing individual parts comprising the interface.

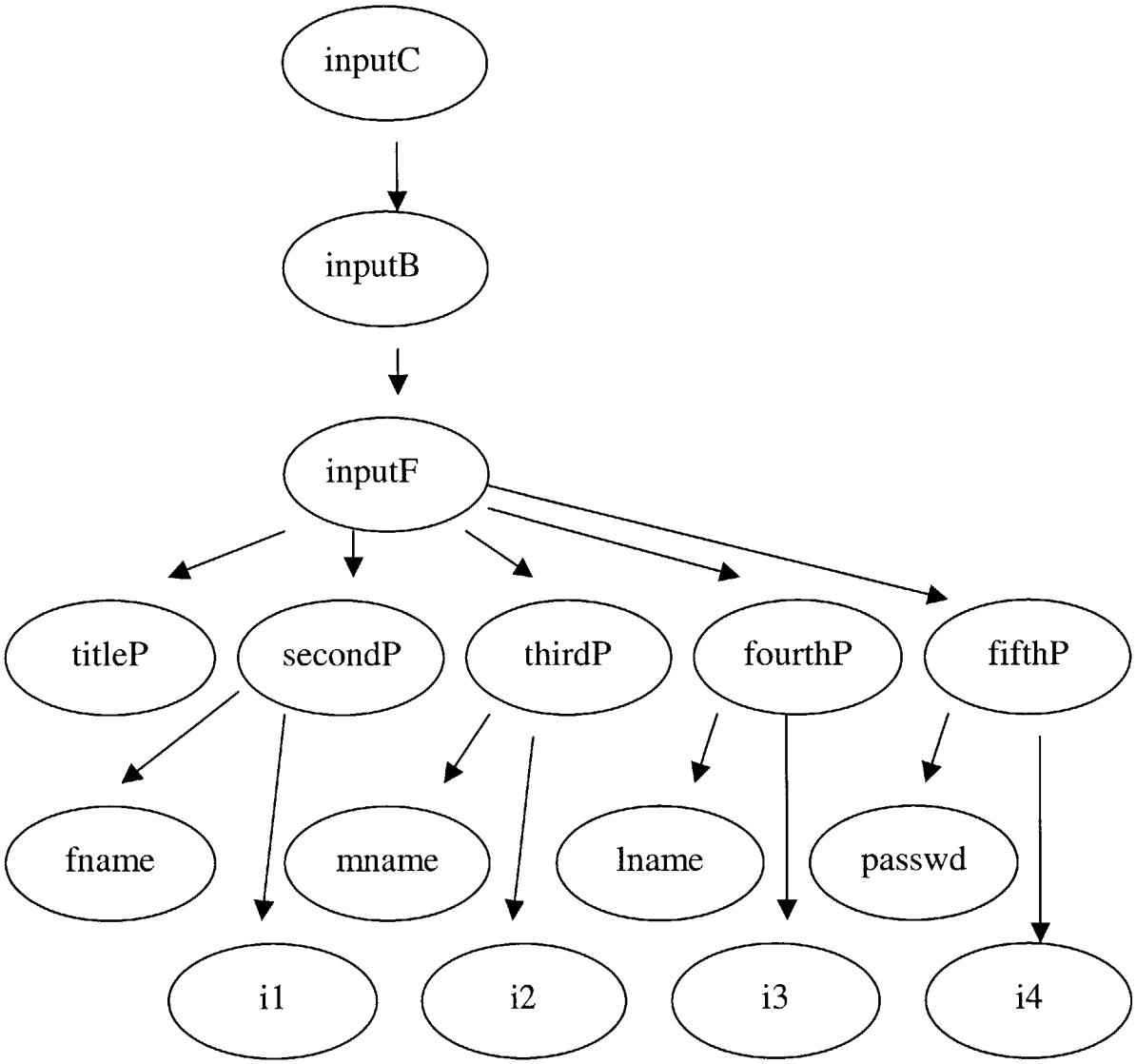


Figure 8 UI Tree

2.9.3 UI Metaphor for HTML

UIML is vocabulary driven. The vocabulary for the target language's UI toolkit is expressed in the peers section of the UIML document. This metaphor or vocabulary is defined by an individual or an organization and stored at an accessible location in the World Wide Web. Individual UIML files using Uniform Resource Indentifier source the peers UIML file.

```
<peers name="HTML">
 <presentation name="HTML">
  <d-class name="Container" maps-type="tag" maps-to="html:HTML"/>
  <d-class name="Body" maps-type="tag" maps-to="html:BODY"/>
  <d-class name="Paragraph" maps-type="tag" maps-to="html:P"/>
  <d-class name="Form"  maps-type="tag" maps-to="html:FORM">
  <d-class name="Text"  maps-type="hidden" maps-to="Nothing">
  <d-class name="Input"  maps-type="tag" maps-to="html:INPUT">
    <d-property name="type"   maps-type="attribute" maps-
to="TYPE">
      <d-param type="String"/>
    </d-property>
    <d-property name="value"   maps-type="attribute" maps-
to="VALUE">
        <d-param type="String"/>
    </d-property>
  </d-class>
 </presentation>
<logic name="HTML>
  <d-component name="form" maps-to="form">
   <d-method name="submit" maps-to="ACTION"/>
    <d-param name="url" type="String"/>
    <d-param name="method" type="String"/>
   </d-method>
  </d-component>
</logic>
</peers>
```

2.9.4 UI Metaphor for WML

Similarly the vocabulary is devised for WML. The peers UIML file for WML is shown below.

```
<peers name="WML">
 <presentation name="WML">
  <d-class name="Container" maps-type="tag" maps-to="wml:WML"/>
  <d-class name="Body" maps-type="tag" maps-to="wml:DECK"/>
  <d-class name="Paragraph" maps-type="tag" maps-to="wml:P"/>
  <d-class name="Form"  maps-type="tag" maps-to="wml:CARD">
  <d-class name="Text"  maps-type="hidden" maps-to="Nothing">
  <d-class name="Input"  maps-type="tag" maps-to="html:INPUT">
    <d-property name="type"   maps-type="attribute" maps-
to="TYPE">
      <d-param type="String"/>
    </d-property>
    <d-property name="value"   maps-type="attribute" maps-
to="VALUE">
        <d-param type="String"/>
    </d-property>
  </d-class>
 </presentation>
 <logic name="WML>
  <d-component name="form" maps-to="CARD">
   <d-method name="submit" maps-to="DO"/>
    <d-param name="url" type="String" maps-to="href"/>
    <d-param name="method" type="String" maps-to="method"/>
   </d-method>
  </d-component>
 </logic>
</peers>
```

2.9.5 Target Code and Interface Generated for HTML

Using the UIML file specified in section 2.9.1 and UI Metaphor specified in 2.9.3 the translator tool translates the UIML file to HTML code as shown below.

```
<HTML>
      <BODY>
            <FORM METHOD="get" ACTION="http://somewhere.com">
                  <p>
                        Log into site:
                  </P>
                  <P>
                        First Name:
                        <INPUT VALUE="Donald"/>
                  </p>
                  <P>
                        Middle initial:
                        <INPUT VALUE="D"/>
                  </P>
                  <p>
                        Last Name:
                        <INPUT VALUE="Duck"/>
                  </p>
                  <p>
                        Password:
                        <INPUT TYPE="PASSWORD" VALUE="Mickey"/>
                  </P>
            </FORM>
      </BODY>
</HTML>
```

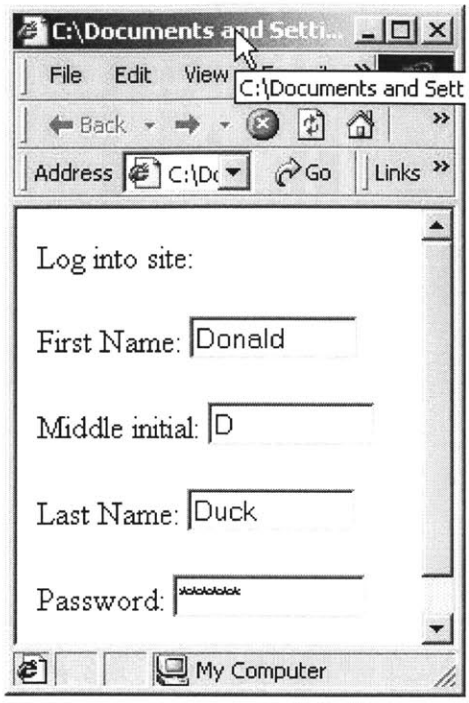
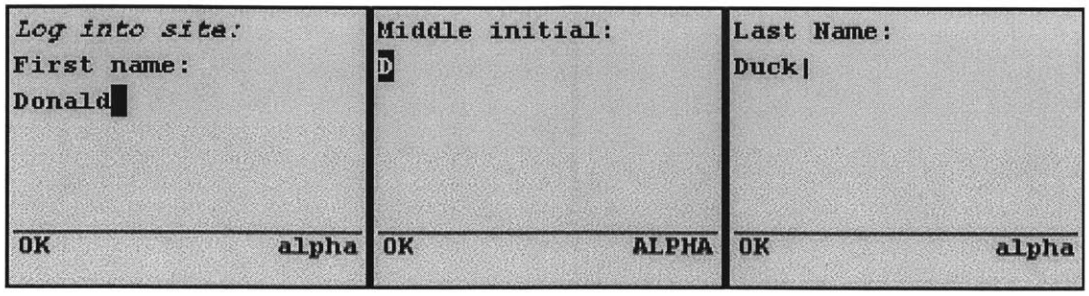


Figure 9 HTML Interface

2.9.6 Target Code and Interface generated for WML

Using the UIML file specified in section 2.9.1 and UI Metaphor specified in 2.9.4 the translator tool translates the UIML file to WML code as shown below.

```
<wml>
     <deck>
          <card>
               <p>
                    Log into site:
               </p>
               <p>
                    First name:
                    <input name="i1" value="Donald"/>
               </p>
               <p>
                    Middle initial:
                    <input name="i2" value="D" />
               </p>
               <p>
                    Last Name:
                    <input name="i3" value="Duck"/>
               </p>
               <p>
                    Password:
                    <input name="i4" type="password" />
               </p>
               <do type="accept">
                <go method="get" href="http://somewhere.com">
               </do>
          </card>
     <deck>
</wml>
```



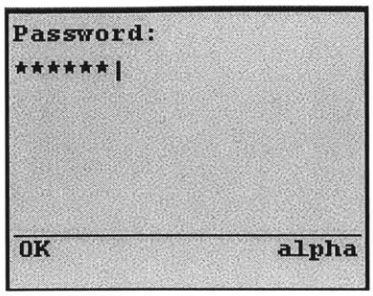


Figure 10 WML Interface

3 Survey of Existing Methodologies and Applications

3.1 Overview

A number of multimodal applications were built and methodologies proposed over the past few years. The focus of this chapter is to survey and analyze them.

3.2 Various Architectures for Multimodal Systems

The main architectural approaches for interpreting multimodal speech, pen-based gestures and keypad input are briefly described below.

3.2.1 Early Fusion or Feature Fusion

In Early fusion architecture, multimodal systems handle joint processing of input signals and integrate signals at the *feature level.* In this architecture, the recognition process in one mode influences the recognition in the other. This is more appropriate for closely coupled modalities and tend to generalize the content and characteristics of the modes, where as in modes that provide complementary information modes should be integrated at utterance level. Hence this architecture is useful only for closely couple modalities. Also some disadvantages associated with this approach are computational intensity and difficulty in training the system.

3.2.2 Late Fusion or Semantic fusion

Multimodal systems based on late fusion architecture integrate information at semantic level. The modal integration is a sequential process where individual recognizers are trained using unimodal data. This type of architecture is easy to implement, as data for unimodal systems is available. Also these systems are easy to scale up with new input modes and vocabulary sets. Semantic fusion architecture is particularly attractive where the modes are complementary and strengths of each mode are used to overcome the weaknesses in the other. This results in a more reliable system due to mutual disambiguation

3.2.3 Multi-agent Architectures

Figure 11 depicts the information flow in a typical multimodal system. Similar to Semantic fusion architecture, various modes are recognized in parallel, and each is processed by an understanding component. The results are meaning representations that are fused by the multimodal integration component, which also is influenced by the system's dialogue management and interpretation of current context. An example of the meaning representation is presented in the next section as part of the case study. The multimodal interpretation or meaning is sent to the application for desired action.

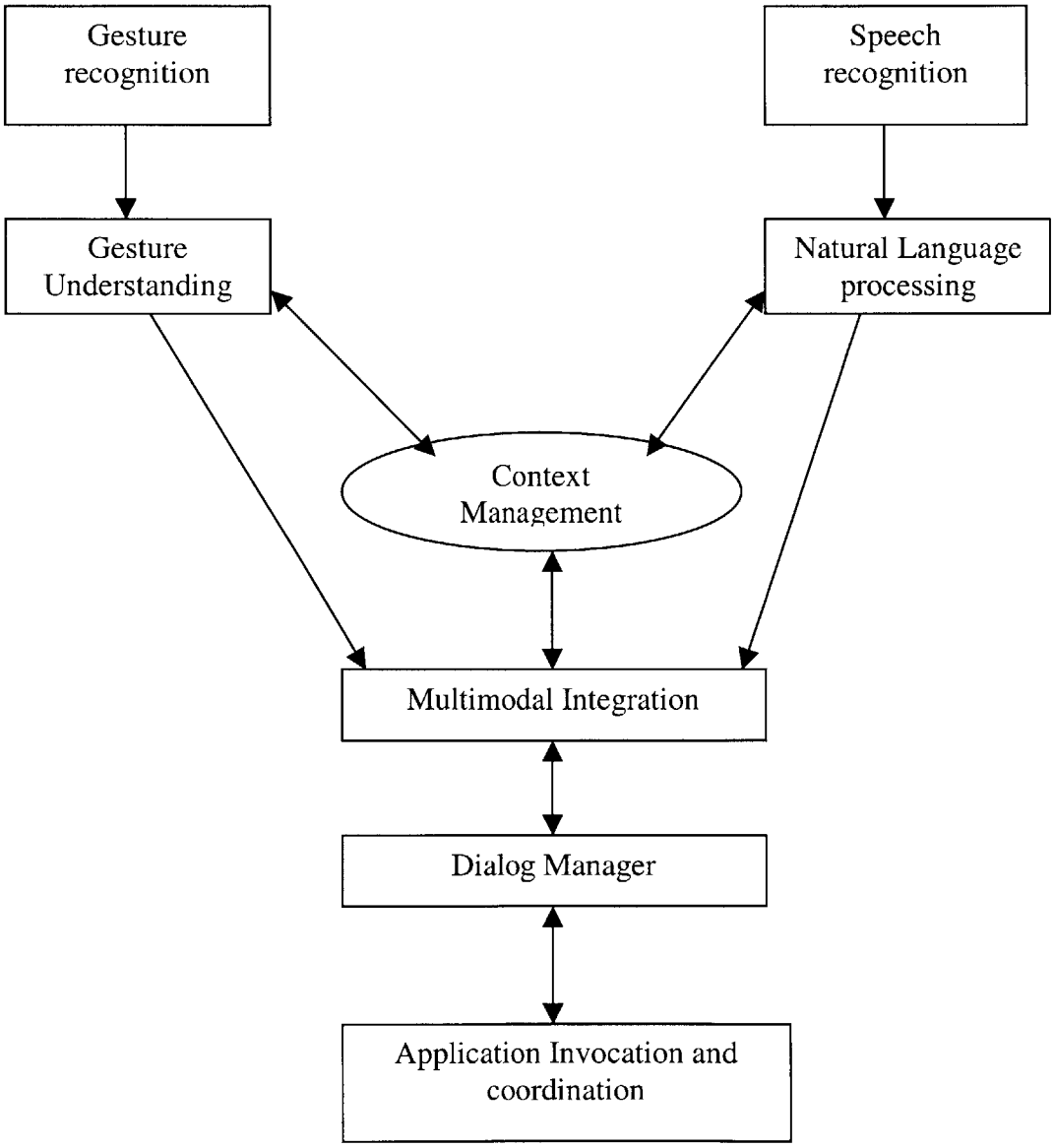


Figure 11 Information Flow in a Multimodal System

There are numerous ways to realize the above information processing flow as architecture. If the system is homogeneous in its programming language, procedule calls or remote procedure calls could be used. However, if the system is heterogeneous (e.g., in programming languages, operating systems, or machine characteristics), the above method may prove difficult. To provide a higher-level layer that supports distributed heterogeneous software, while shielding the designer from the details of communication, each component is "wrapped" by a layer of software that enables it to communicate via a standard language over TCP/IP. This design has the advantage of no intermediaries, but it can be brittle in the face of agent failure.

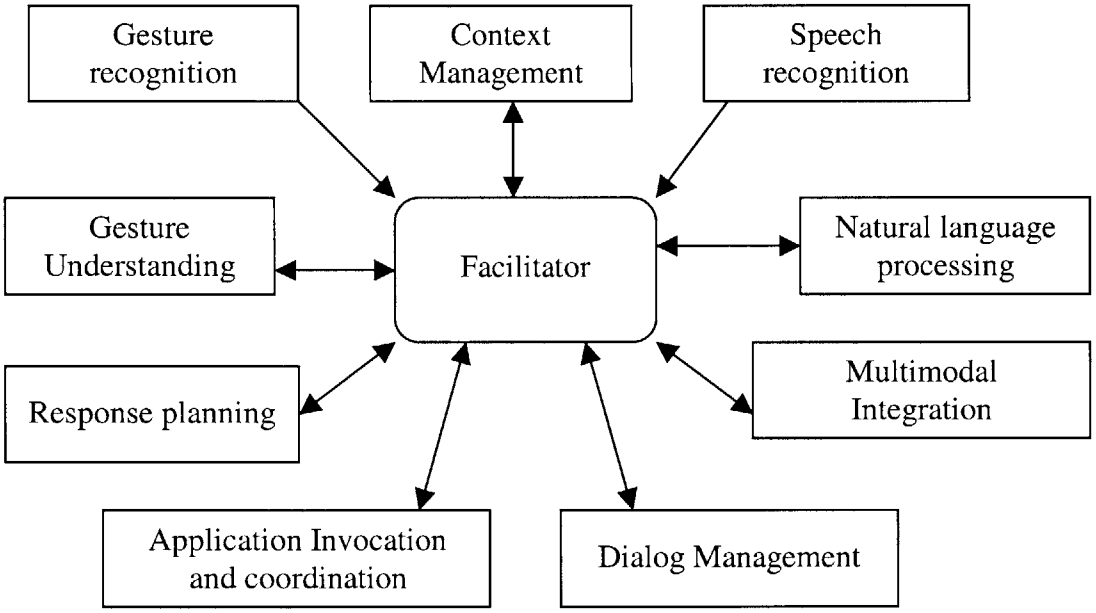


Figure 12 Open Agent Architecture for Multimodal Integration

As an alternative design, many architectures have adopted a *facilitated* form of communication, in which agents do not need to know to whom they are making requests or supplying answers. Instead, these agents communicate through a known facilitator, which routes messages to the interested and capable receivers. This becomes a *hub-and-spoke architecture,* with all agents communicating via the central facilitator. The facilitator provides a place for new agents to connect at run time, and they then can be discovered by other agents and incorporated into the ongoing distributed computation.

The hub also becomes a locus for building collaboration systems, since the facilitator can route communications to multiple agents that may be interested in the same messages.

3.3 MMI System – A Case Study

3.3.1 Introduction and Purpose

The MMI System is a Man-machine Interface for MultiModal Interaction with knowledge-based systems (KBS). The objective of the MMI project was to develop a toolkit and methodology to construct Knowledge based systems (KBS) in different domains, which have multimodal interfaces and undertake cooperative dialogue with users. The focus of the system is on "multimodal" rather than "multimedia" interaction in the interface since a multi-media system is one, which uses different presentation media (e.g. text, raster graphics, video, speech) without a commitment to the underlying representation of the information processed. The system includes several input and output media, but is committed to a single internal representation for the information. The modes available in the MMI system are: for input: English, French and Spanish natural languages, gesture, direct manipulation of graphics, command language; and for output: English, French and Spanish natural languages, graphics (CAD diagrams and business graphics), and non-verbal audio. The representation language used for all information within the system is Common Meaning Representation (CMR).

3.3.2 System Architecture

The architecture of the MMI system can be described as three layers; the top layer contains the input and output presentation modes, the middle layer is the dialogue management layer, and the bottom layer is the application knowledge based system. Expert modules exist within each of the three layers. These expert modules are simple modules performing specific tasks with their own private data structures and allow a sufficiently coherent set of processes to be gathered in a single module.

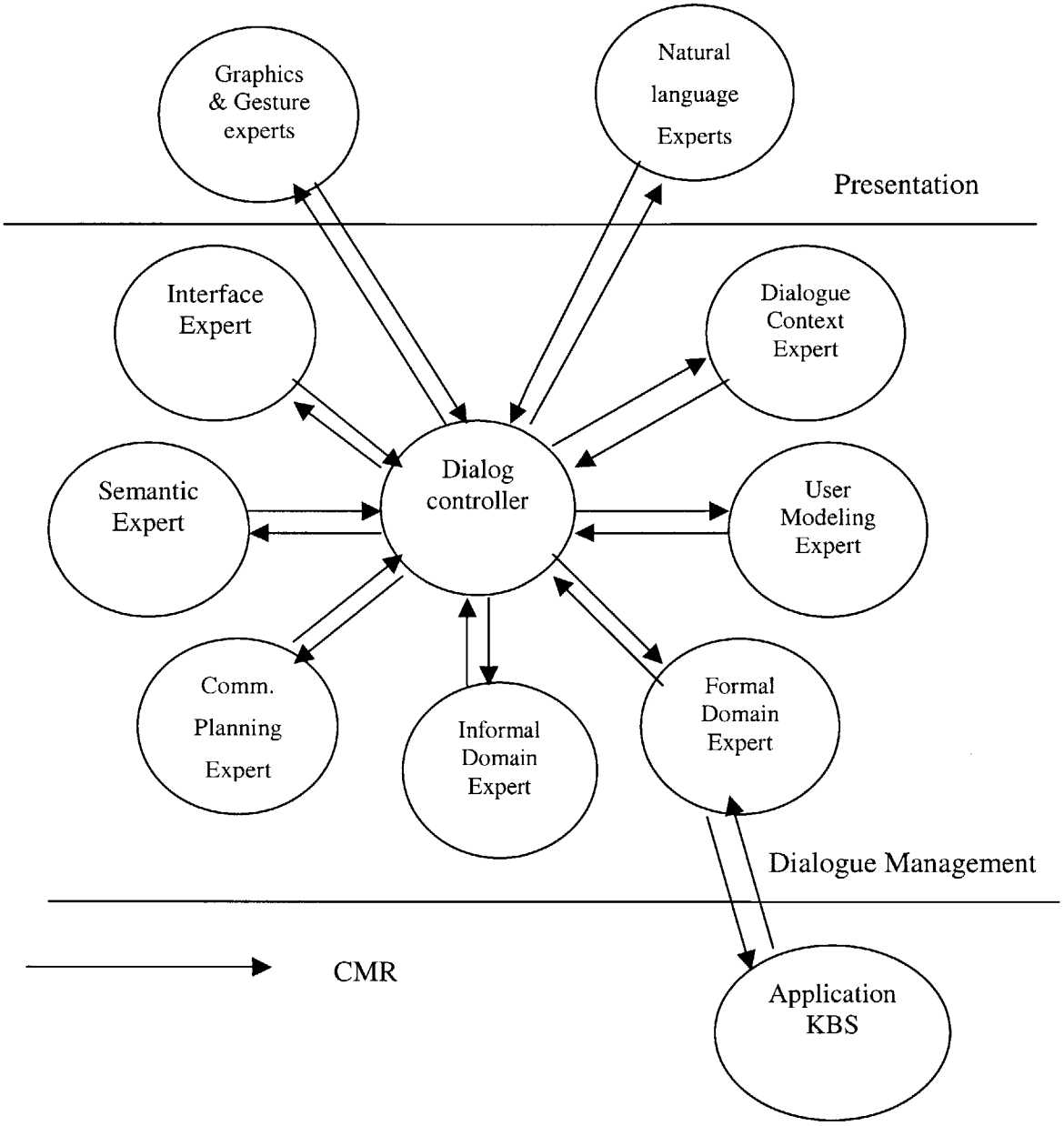


Figure 13 Architecture of MMI System

All operations within the dialogue management layer are performed on a common meaning representation, which is "independent" of both the application and any specific

mode. All input from the modes to the dialogue management is cast in the common meaning representation, as is any output to the modes. This was done considering that the architecture should be portable between applications. Dialog management contains a module that maps common meaning representation onto the application language.

Below is the description of all expert modules present and their respective responsibilities.

The **dialogue controller** deals with: major control loop to manage the structure of the dialog; activating whatever experts are necessary to support dialogue function.

The **dialogue context expert** manages every thing, which has to with recording the dialog structure and extracting relevant information from it.

The **user modeling expert** extracts user beliefs and preferences from CMR information; stores the user model and provides access to the user model.

The **informal domain expert** is responsible for storing plans for the domain task evaluating informal aspects of the dialog.

The **communication planning expert** is responsible for constructing the logical form of the system output selecting the output mode.

The **formal knowledge expert** is responsible for translating the CMR into domain terms representing metaknowledge of the domain.

The **interface expert** is responsible for translating the CMR into interface terms the current physical layout of the interface.

The **semantic expert** has all the knowledge about the international definitions of the predicates used in CMR.

In the **mode layer** the various mode experts are responsible for input and output of each mode to/from CMR.

3.3.3 Methodology

The application uses a frame architecture with constraints on the slots for objects in the design. Initially the user describes some requirements; the expert will then elicit more detailed requirements; then an initial design is produced; the user will then investigate the design and suggest modifications to both the design and the requirements; a new design is produced and investigated. This cycle is repeated several times.

Since the interactions could be expressed equivalently in several modes, the common meaning representation formalism, common to all nodes is used as a vehicle for internal communication of the semantic content of the interactions inside the interface and also used for semantic and pragmatic reasoning. The meaning representation formalism is called the CMR (Common Meaning Representation). The purpose of the CMR is to represent the meaning of interactions between the system and the user, or the user and the system. Such interactions are called "communication actions". In MMI a communication action is a graphical action, a command language/gesture action, or a natural language action, which can be performed by either the user or the system. When the communication action is expressed in a natural language, the action is an utterance. A CMR expression contains four sorts of information: utterance type, annotations, logical formula, syntactic information. When a user interacts with the system, the interaction is encoded as a CMR expression. This CMR expression is used by the Dialog control to query or exchange information with other expert modules like the user modeling expert, semantic expert, mode expert, communication planning expert etc. Finally, after extracting the semantic information from the CMR, a cooperative response is provided to the user.

3.3.4 Analysis

The MMI system is an ambitious architecture for a multimodal interface system. All the features were not developed to the same degree of completeness. The creators of the system are not certain about how much effort it would take to build similar systems for various domains. However, the architecture itself is a clean and modular way for implementation of multimodal systems. The important thing to note from the system is the concept of common meaning representation formalism. In today's information age, there is a desire to build multimodal interfaces for every application developed. Adopting the MMI architecture is not only difficult to implement but would be hard to maintain. With the inception of the World Wide Web all the content and data are stored in XML format. My belief is that it is possible to replicate a XML representation of CMR plus the functionality of some or all the features of the various expert modules mentioned. This would enable easy authoring of multimodal applications and where the XML

representation cannot satisfy all the requirements of the Multimodal dialog language, expert modules could be introduced. For instance, from the MMI architecture, the natural language experts could use the representation of knowledge (authors intent) in Natural language Markup language. The Graphics and Gesture experts use the information authored in various UI markup languages like HTML, WML, and VoiceXML. User modeling experts can use information from the CC/PP guidelines laid down by W3C. Later in chapter 5 we will see that instead of having various XML languages for different expert modules, one language – UIML could be used as a unified equivalent representation of CMR or all the XML languages mention above.

4 Methodology

4.1 Overview

In chapter 2 it was seen that UIML makes it easier to build UI applications for various platforms and unifies the cross-platform application development process. In chapter 3, various multimodal applications, methodologies and architectures are inspected. In this chapter a simple methodology is proposed to build multimodal interfaces using UIML for platforms available today. The languages like HTML, WML, VoiceXML, Java Swing/AWT, and Palm API are treated loosely as modes available as these are the popular languages in which most of the user interface development is done today. For simplicity and efficient communication of the concept, two modes, WML (for display) and VoiceXML (for Voice) are chosen for demonstration of concepts through examples.

4.2 Prolog

Today we do not have a multimodal dialog language that takes into consideration all the capabilities of a device or platform and that does mode integration. Even if we have one in the near future, browsers will have to be built to render the applications written in this language. Though this will eventually be a reality, can we solve the problem of mode integration using UIML with the existing Dialog Markup languages like VoiceXML and WML for which individual browsers exists? Solving this problem involves various steps. Firstly, considering that both VoiceXML and WML are dialog markup languages, it is possible to build a common or generic vocabulary for VoiceXML and WML. For instance a class *Dialog* could represent both classes *Form* (VoiceXML) and *Card* (WML), *Menu* for *Menu* (VoiceXML) and *Select*, *Input* for *Field* (VoiceXML) and *Input, Choice* for *Choice* (VoiceXML) and *Option, output* for *Speech* (VoiceXML) and *Text* and similarly for other classes common vocabulary could be designed. In case of UI classes that are unique to that modality or language, they have separate class names. For instance *Prompt* and *Enumerate* in VoiceXML. There will be two <peers> sections named WML and Voice, each to map common or generic vocabulary to target specific classes. Writing common vocabulary serves two purposes

- Identifying functional equivalent markup of the application in each language.
- Secondly not having to write the same part in the interface twice.

For example,

<part name="get_users_choice" class="Menu" mode="WML Voice"> makes it clear that the part is used in both the WML and Voice interfaces. The interpreter or compiler maps the part to the corresponding <peers> sections -<peers name="WML"> and <peers name="VoiceXML">. Though the mode attribute is not part of UIML2, it or an equivalent attribute will be part of the language soon.

4.3 Proposed Architecture

Figure 14 shows the architecture in which there is a standard Multi-modal Dialog Markup Language and rendering is done based on the vocabulary defined for the Multimodal Dialog Markup Language. The Multimodal Dialog Markup Language specifies mode integration, synchronization of input and output modes etc. In a situation where the Multimodal Dialog Markup Language does not exist, an Interface agent/Multimodal dialog manager is introduced into the architecture for the purposes of mode integration, synchronization of input and output modes. This is loosely based on the Multi-agent architecture discussed in section 3.2.3. The reason for choosing the Multi-agent architecture is that it is modular and extensible for additional tasks like semantic integration, user modeling etc. Figure 15 shows this kind of an architecture.

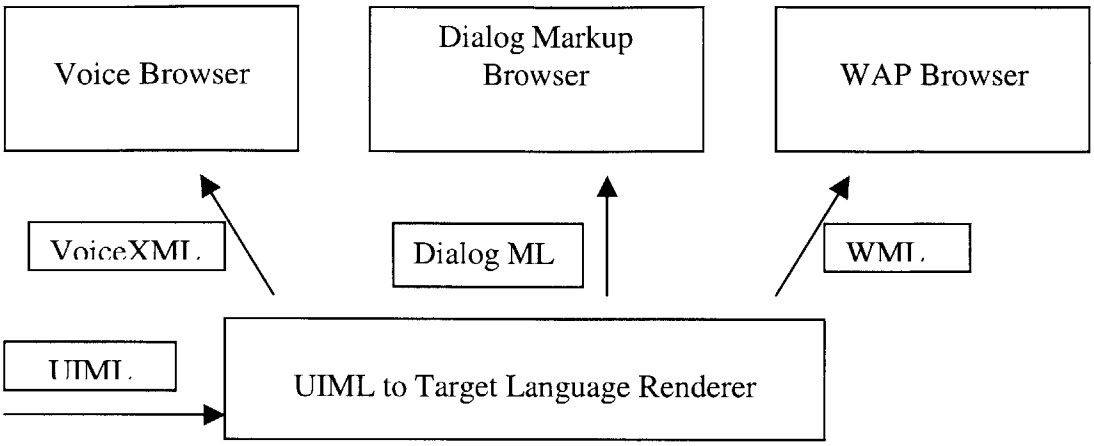


Figure 14 Architecture for Rendering Target Language Interfaces

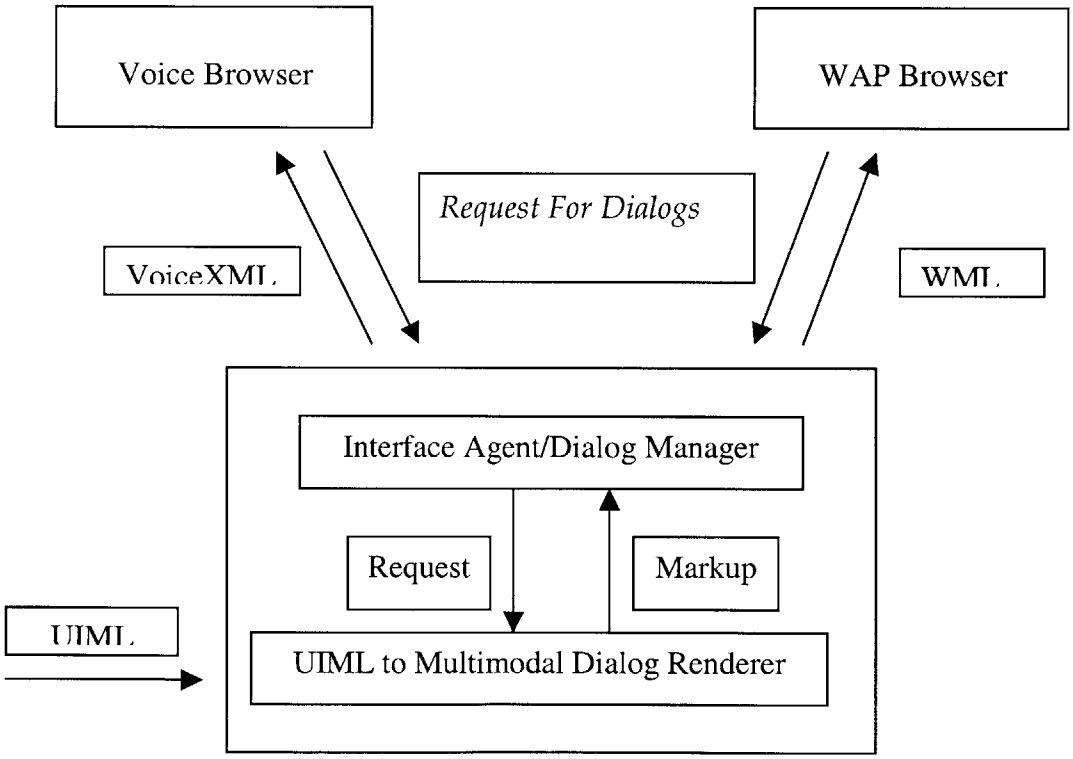


Figure 15 Architecture for Rendering Multimodal Interfaces

4.4 Proposed Methodology

When the UIML is initially given as input to the renderer, it builds a virtual UI tree with parts represented by nodes. Each of the nodes has a property "mode" identifying the mode it is associated with in addition to the other properties specified in the style section of the UIML file. Synchronization points are identified to be the nodes in the UI tree for which a behavior rule is specified. The entire UI tree is subdivided into UI sub-trees, sub division done at the synchronization points. Typically the synchronization points are associated with input UI widgets or individual dialogs. Additionally a rule that specifies the communication with the interface agent is added to the parent node in each of the sub-trees. This is done by modifying the Document Object Model's API.

The Interface Agent coordinates the dialogs in input and output modes based on the rules programmed in it. After making a decision, the agent requests the renderer for equivalent target language code for any of the UI sub-trees. The user could also specify the preference of modality as a parameter to the Multimodal Interface Agent. Then an attempt is made to present the entire interface in that mode. This is useful taking accessibility into consideration. The resulting VoiceXML and WML code is presented to the Voice browser and WML browser.

The Interface Agent can be programmed for a plethora of tasks. It could handle user modeling by maintaining user profiles in session and adapting the interface. In the example presented in the next section the Interface Agent was programmed for two tasks and the necessary knowledge was stored in the agent. The tasks are to present the interface in the preferred modes, handle recognition errors and ambiguities, and to synchronize various input modes and output modes (mode integration).

4.5 Example

Lets consider an example to illustrate the above methodology for an application that does mode integration and adapts itself to recognition errors. A single dialog is examined in a network management example where the user is presented with a menu of printers to select from. The dialog is prone to voice recognition errors, as the author of the UI is not sure if the printer names are part of the voice recognizer's dictionary. Hence he specifies

the interface in both Voice and display modes, though the preferred mode is voice. Also, if the user does not prefer voice and would like only a display mode, he could specify so to the Multimodal interface agent. The multimodal user interface in UIML is written as

```
<part name="select"   class="Menu" vocabulary="WML  Voice">
   <part name="welcome" class="Prompt" vocabulary="Voice">
      <part name="say" class="Speech"  vocabulary="Voice"/>
      <part name="oneof" class="Enumerate" vocabulary="Voice"/>
   </part>
   <part name="printer1" class="Choice" vocabulary="WML  Voice">
      <part name="p1" class="Output" vocabulary="WML  Voice"/>
   </part>
   <part name="printer2" class="Choice" vocabulary="WML  Voice">
      <part name="p2" class="Output" vocabulary="WML  Voice"/>
   </part>
</part>
```

The of <style> section is

```
<property name="content" part-name="say">
     Welcome. Say one of:
</property>
<property name="next" part-name="printer1">
     http://www.printer1.com/start.uiml
</property>
<property name="content" part-name="p1">
     HP-LaserJet 5M
</property>
<property name="next" part-name="printer2">
     http://www.printer2.com/start.uiml
</property>
<property name="content" part-name="p1">
     Lexmark Optra-E310
</property>
```

If the platform supports only voice (device is equipped with Voice browser) only or the user prefers the voice mode only, then the interface agent requests the renderer to render the dialog only in voice mode and the resultant equivalent interface in VoiceXML will be

```
<menu>
   <prompt>Welcome. Say one of:<enumerate/></prompt>
   <choice next="http://www.printer1.com/start.uiml">
```

```
      HP-LaserJet 5M
   </choice>
   <choice next="http://www.printer2.com/start.uiml">
      Lexmark Optra-E310
   </choice>
</menu>
```

If the platform supports display (equipped with WAP browser) only or the user prefers the display mode only, then the interface agent requests the renderer to render the dialog in display mode and the resultant equivalent interface in WML will be

```
<select>
   <option onpick=" http://www.printer1.com/start.uiml ">
      HP-LaserJet 5M
   </option>
   <option onpick=" http://www.printer2.com/start.uiml ">
      Lexmark Optra-E310
   </option>
</select>
```

It is observed from above example that an interface can be written for multiple platforms in a more compact and natural way when the functional similarities are quite significant in an interface for different platforms. The initial behavior of the interface in UIML is given below.

```
<rule>
   <condition>
      <event class="Nomatch" part-name="select"/>
   </condition>
   <action>
      <restructure    at-part="select"    how="cascade"    where-
part="last">
         <template>
            <structure>
               <part name="please" class="Speech">
                  <style>
                     <property  name="content"  >Please  say  one
of</property>
                   </style>
               </part>
               <part name="again"     class="Enumerate"/>
```

```
            </structure>
          </template>
        </restructure>
    </action>
</rule>
```

The representation of synchronization points chosen in the presented example is shown in figure 16. At time = 0, the Interface Agent chooses the most usually preferred mode i.e voice to render the initial interface. The user could also specify the preference as a parameter to the Renderer. The Renderer emits VoiceXML and is presented to the Voice browser.

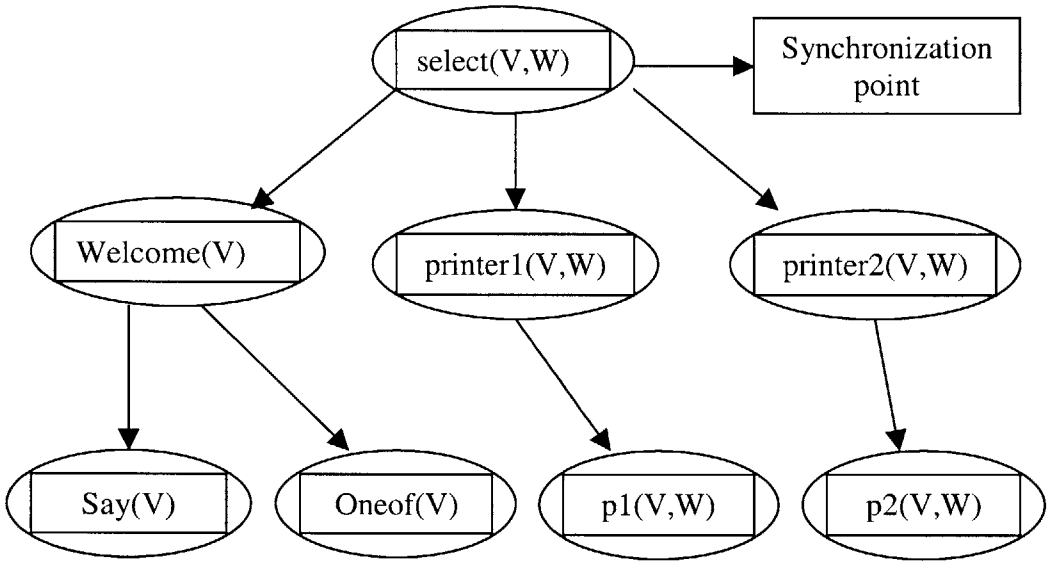


Figure 16 UI tree Representation of Initial Interface

V – VoiceXML
W - WML

As noticed the choice presented in the menu is difficult for text to speech synthesis, and it is possible that the user might not understand the choice or the browser does not recognize the option specified by the user. In either situation, a no match event is fired and the Interface Agent is notified. The Interface Agent decides (rule based dialog management) that it is appropriate to give the user a second chance and hence requests the Renderer to present the appropriate interface in the voice mode. The Renderer

extracts the UI sub tree and presents the target Voice markup to the Agent who in turn presents it to the voice browser. If a similar situation is repeated, the Interface Agent requests the Renderer to present the interface in the alternate mode. It could be because the author specifies explicitly in the behavior section or a decision is taken by the agent based on the knowledge it posses. Renderer presents the interface in WML from the node for which the synchronization point was identified. For Example

```
<rule>
   <condition>
      <event class="Nomatch" part-name="select"/>
   </condition>
   <action>
       Instruction to switch modes.
   </action>
</rule>
```

The above rule is not part of the original interface developed by the author, but a result of modification of the UI tree by the interface agent. If the user asks for help, the rule such as the one specified will transfer the control to the dialog specified in the action, facilitating simultaneous input.

```
<rule>
   <condition>
      <event class="Help" part-name="select"/>
   </condition>
   <action>
      <call name="form.goto">
           <param name="next">
                www.help.printer.com/help.uiml
           </param>
      </call>
   </action>
</rule>
```

Different states of the UI tree in the dialog are shown below:

State 1. Only Voice interface exists

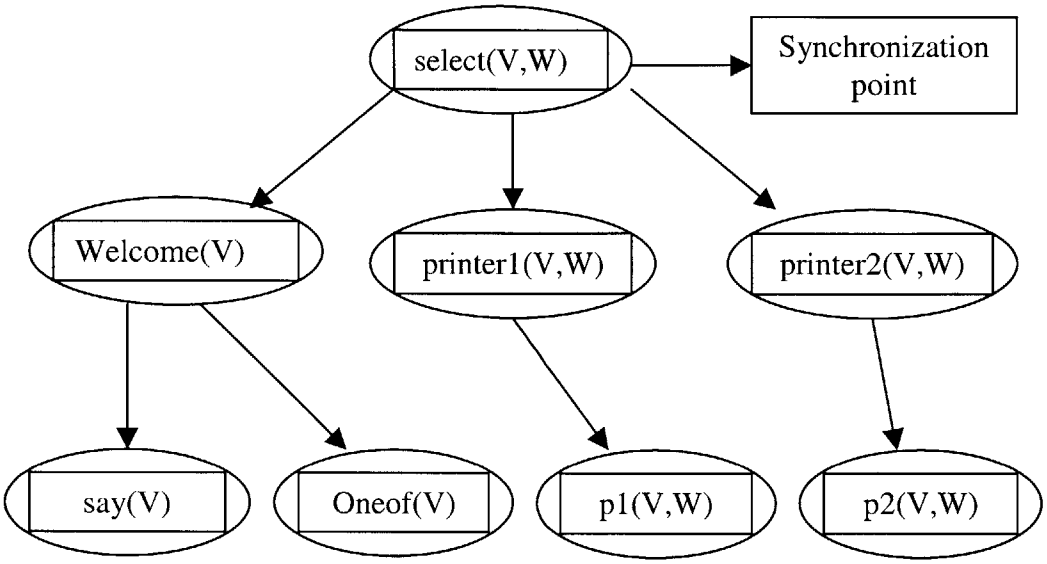


Figure 17 State 1 of the Dialog

State 2. When a Nomatch event is fired, User is given the second chance to present his choice

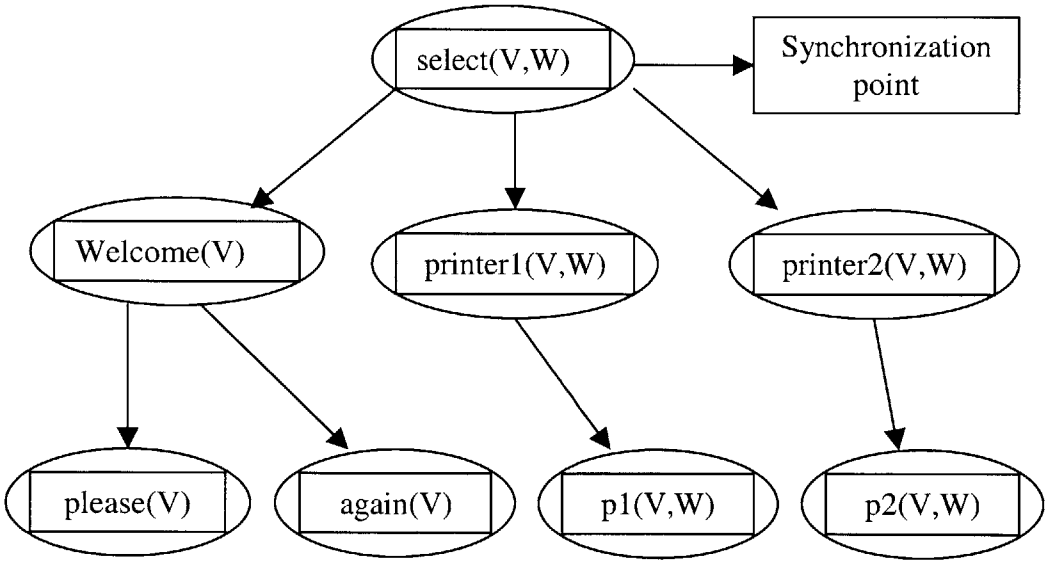


Figure 18 State 2 of the Dialog

State 3. When the user is unsuccessful is specifying his choice, the mode is switched. But the synchronization is preserved and so the voice browser is presented with the empty menu with a rule specifying that if user asks for help, transfer the control to a different dialog.

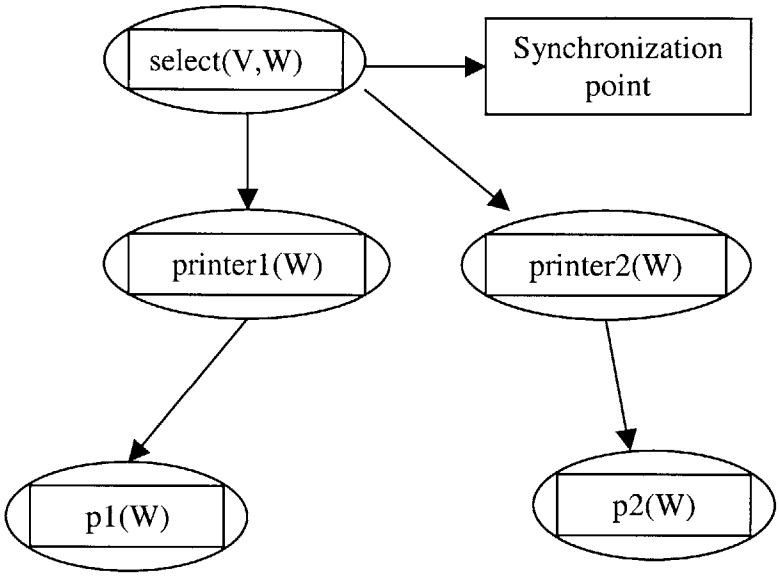


Figure 19 State 3 of the Dialog

4.6 W3C's Requirements for Authoring Multimodal Dialogs

In the previous section a simple methodology and architecture was proposed for mode integration in UIML using the existing markup Languages like WML and VoiceXML. Below we provide the various requirements W3C has laid down for authoring multimodal dialogs and a discussion to illustrate with examples how UIML satisfies the major set of those requirements.

4.6.1 General Requirements

4.6.1.1 Scalable across end user devices

Requirement:

"The markup language will be scalable across devices with a range of capabilities, in order to sufficiently meet the needs of consumer and device control applications. This includes devices capable of supporting:

- Audio I/O plus keypad input - e.g. the plain phone with speech plus dtmf, MP3 player with speech input and output and with cellular connection to the Web;
- Audio, keypad and small screen - e.g. WAP phones, smart phones with displays;
- Audio, soft keyboard, small screen and pointing - e.g. palm-top personal organizers with cellular connection to the Web.
- Audio, keyboard, full screen and pointing - e.g. desktop PC, information kiosk.

The server must be able to get access to client capabilities and the user's personal preferences" – W3C

Conformance in UIML:

UIML was designed precisely for the requirement specified above. UIML is extensible. Extensible meaning that the language is not tied with one toolkit or API but only specifies the language syntax and allows integration of any API or toolkit resources with the language. It is truly cross-platform and scalable across devices with different capabilities. Much of the discussion to this cause is presented in chapter 2.

4.6.1.2 Easy to implement

Requirement:

"The markup language should be easy for designers to understand and author without special tools or knowledge of vendor technology or protocols (multimodal dialog design knowledge is still essential)." – W3C

Conformance in UIML:

UIML promotes the best user interface design policies by separating the content and presentation from logic. Each of the components could be specified different individuals or organization. For example, an organization maps the generic widget sets to the platform specific widget sets i.e. present a UI metaphor. A service provider or back-end application developer creates the logic section that specifies the application logic. A

content provider could provide content in XML format or use models. Finally the UI designer can integrate all of the above with ease in UIML and concentrate on the UI design in interface section rather than worry about the target toolkit or content or application logic. Multimodal dialog design is still essential to specify the modes that go with a UI part and UI style in the interface.

4.6.1.3 Complementary Use of Modalities

Requirement:

"A characteristic of speech input is that it can be very efficient - for example, in a device with a small display and keypad, speech can bypass multiple layers of menus. A characteristic of speech output is its serial nature, which can make it a long-winded way of presenting information that could be quickly browsed on a display.
The markup will allow an author to use the different characteristics of the modalities in the most appropriate way for the application." – W3C

Conformance in UIML:

As seen in section 4.5, the example illustrates how the author can specify different modalities that can be used for different user interface tasks to exploit various characteristics of the modalities. In the network management example from section 4.5, the options in the menu could be presented only in the display mode and accept the user's selection either from display if users selects it in display and in voice if the user prefers to speak instead. Usually display is preferred for output and voice for input. To avoid recognition of errors, both input and output could be presented in all the available modes though it is not desirable. This would create confusion but at the same time a good practice for maintaining redundancy to avoid errors or ambiguities.

4.6.1.3.1 Output Media

Requirement:

"In a speech plus GUI system, the author will be able to choose different text for simultaneous verbal and visual outputs. For example, a list of options may be presented

on screen and simultaneous speech output does not necessarily repeat them (which is long-winded) but can summarize them or present an instruction or warning.

The markup language will allow speech output to have different content to that of simultaneous output from other media." – W3C

Conformance in UIML:

Consider the example below in which a part "output" is available in modes - display (HTML, WML) and voice (VoiceXML) but the property 'content' of these parts are different in those modes.

```
<interface>
    <structure>
        <part name="present" class="Output" mode="HTML Voice"/>
    </structure>
    <style>
       <property name="content" part-name="present" mode="HTML">
            This site is an e-commerce site. In order to protect
user privacy and provide security, users have to be authenticated
with login and password. . . . . Please enter the password
       </property>
       <property name="content" part-name="present" mode="Voice">
            don't forget to logoff
       </property>
       <property name="content" part-name="present" mode="WML">
            Type in your login and password
       </property>
    </style>
</interface>
```

As seen above, since desktop computers have large display size, the content displayed in HTML browsers can be large and descriptive where as in WML browsers the since the display size is small, a concise content can be specified. In voice, a caution message can be played at the same time.

4.6.1.3.2 Input Modalities

Requirement:

"The markup language will allow, in a given dialog state, the set of actions that can be performed using speech input to be different to simultaneous actions that can be performed with other input modalities." – W3C

Conformance in UIML:

Consider the example below, in which the same input UI widget is specified in display and voice mode but the actions taken for these inputs, is different in each mode.

```
<interface>
    <part name="accept" class="input" mode="WML Voice"/>
      . . . . . .
    <part name="present" class="Output" mode="HTML Voice"/>
</interface>
<behavior>
    <rule mode="WML">
       <condition>
          <event class="input_received" part-name="accept"
       </condition>
       <action>
          <property name="content" part-name="present">
               input received in display mode
          </property>
       </action>
    </rule>
    <rule mode="Voice">
       <condition>
          <event class="input_received" part-name="accept"
       </condition>
       <action>
          <property name="content" part-name="present">
               input received in voice mode
          </property>
       </action>
    </rule>
</behavior>
```

In the above example, when the input is received in two different modes, different content is presented to the user in those modes. This was possible by associating the different roles to different modes.

4.6.1.4 Seamless Synchronization of the various Modalities

Requirement:

"The markup will be designed such that an author can write applications where the synchronization of the various modalities is seamless from the user's point of view. That is, a cause in one modality results in a synchronous change in another. For example:

- An end-user selects something using voice and the visual display changes to match;
- An end-user specifies focus with a mouse and enters the data with voice - the application knows which field the user is talking to and therefore what it might expect; " – W3C

Conformance in UIML:

In UIML, not only the parts that are associated with a particular mode can be specified but also the properties, content and the behavior. Consider the example below.

```
<interface>
    <part name="accept" class="Button" mode="HTML">
      ..........
    <part name="read" class="Speech" mode="Voice">
</interface>
<behavior>
    <rule>
       <condition>
          <event class="ButtonPressed" part-name="accept"
       </condition>
       <action>
          <property name="content" part-name="read">
               Button pressed in display mode
          </property>
       </action>
    </rule>
```

```
</behavior>
```

In the UIML fragment above, the author specifies that when an event "ButtonPressed" event occurs in HTML browser, the content 'Button pressed in display mode" be read to the user. This kind of synchronization in UIML is natural and seamless from user's point of view.

4.6.1.5 Multilingual and International Rendering

4.6.1.5.1 One Language per document

Requirement:

"The markup language will provide the ability to mark the language of a document."

Conformance in UIML:

In UIML the content is separated from other sections of the interface. Hence it is possible to build interfaces with content in different languages as shown in section 2.5.2.8

4.6.1.5.2 Multiple Languages in the same document

Requirement:

"The markup language will support rendering of multi-lingual documents - i.e. where there is a mixed-language document. For example, English and French speech output and/or input can appear in the same document - a spoken system response can be "John read the book entitled 'Viva La France'."" – W3C

Conformance in UIML:

The language content could be specified in different content sections of the same document as

```
<interface>
    <content name="english"> what is your name </content>
    <content name="hindi">  ------------  </content>
    <style>
       <property name="content" part-name="text">
            <content name="English/>
       </property>
    </style>
```

```
</interface>
```

In the style section the required content is referenced.

4.6.2 Input Modality requirements

4.6.2.1 Audio Modality Input

Requirement:

"The markup language can specify which spoken user input is interpreted by the voice browser." –W3C

Conformance in UIML:

Present UIML2 does not provide context information to the browser. Probably a modification has to be proposed for to accommodate such functionality.

4.6.2.2 Sequential multi-modal Input

Requirement:

"The markup language specifies that speech and user input from other modalities is to be interpreted by the browser. There is no requirement that the input modalities are simultaneously active. In a particular dialog state, there is only one input mode available but in the whole interaction more than one input mode is used. Inputs from different modalities are interpreted separately. For example, a browser can interpret speech input in one dialog state and keyboard input in another.

Things like input events define the granularity. Synchronization does not occur at any finer granularity. When the user takes some action, only one mode of input will be available at that time.

Examples:

1. In a bank application accessed via a phone, the browser renders the speech "Speak your name", the user must respond in speech and says "Jack Jones", the browser renders the speech "Using the keypad, enter your pin number", the user must enter the number via the keypad.
2. In an insurance application accessed via a PDA, the browser renders the speech "Please say your postcode", the user must reply in speech and says "BS34 8QZ", the browser renders the speech "I'm having trouble understanding you, please

enter your postcode using the soft keyboard." The user must respond using the soft keyboard (i.e. not in speech). " – W3C

Conformance:

The above requirement specifies that only one mode should be available input. A part in UIML could be used with only one mode associated with it. The example in section 4.5 illustrates that the initial dialog is presented in voice and when the user is unable to communicate in this mode due to recognition errors, the input mode is switched to display as specified by the author in a rule.

4.6.2.3 Input modes supported

Requirement:

"The markup language will support the following input modes, in addition to speech:

- DTMF
- Keyboard
- Pointing device (e.g. mouse, touchscreen, etc) " – W3C

Conformance in UIML:

UIML supports all the modes specified by the target toolkit, API or language.

4.6.2.4 Extensible to new input media types

Requirement:

"The model will be abstract enough so any new or exotic input media (e.g. gesture captured by video) could fit into it."

Conformance:

UIML is extensible for new Languages, platforms, API and modes. Much of the discussion in this respect is provided in chapter 2.

4.6.2.5 Modality-independent representation of the meaning of user input

Requirement:

"The markup language should support a modality-independent method of representing the meaning of user input. This should be annotated with a record of the modality type. The markup language supports the same semantic representation of input from different modalities. For example, in a pizza application, if a topping can be selected from an option list on the screen or by speaking, the same semantic token, e.g. 'topping' can be used to represent the input."

Conformance:

The generic vocabulary presented in section 4.2 illustrates that a common set of UI widgets could be identified, that are common to a set of modes and hence can be used in any of the modes. An author can specify a particular mode by using the attribute (annotation) 'mode'.

4.6.2.6 Coordinate speech grammar with grammar for other input modalities

Requirement:

"The markup language coordinates the grammars for modalities other than speech with speech grammars to avoid duplication of effort in authoring multimodal grammars." – W3C

Conformance in UIML:

The input grammars can be specified in UIML as models. For example,

```
<content>
   <constant name="action" model="one-of">
      <constant model="item" value="open"/>
      <constant model="item" value="close"/>
   </constant>
</content>
```

The model in the above fragment used is 'one-of' with sub models 'item'. Models can be custom built or can be extracted from external XML files or databases. The above model is translated to Grammar Markup Language as

```
<rule id="action">
```

```
    <one-of>
       <item >   open </item>
       <item >  close </item>
     </one-of>
</rule>
```

The same grammar model could be translated into WML grammars as

```
<select>
    <option>Open</option>
    <option>Close</option>
</select>
```

4.6.3 Output Media Requirements

4.6.3.1 Temporal semantics for synchronization of voice input and output with multimedia

Requirement:

"The markup language will have clear temporal semantics so that it can be integrated into the SMIL multimedia framework. Multi-media frameworks are characterized by precise temporal synchronization of output and input. For example, the SMIL notation is based on timing primitives that allow the composition of complex behaviors."

Conformance:

UIML provides synchronization of modalities only at event level. For finer synchronization, UIML can be used with a synchronization language like SMIL

4.6.3.2 Display size

Requirement:

"Visual output will be renderable on displays of different sizes. This should be by using standard visual markup languages e.g., HTML, CHTML, WML, where appropriate. This requirement applies to two kinds of visual markup:

- Markup that can be rendered flexibly as the display size changes
- Markup that is pre-configured for a particular display size. " – W3C

Conformance in UIML:

Refer to section 4.6.4.1

4.6.4 Architecture, Integration and Synchronization points

4.6.4.1 Reuse standard markup languages

Requirement:

"Where possible, the specification must reuse standard visual, multimedia and aural markup languages, including:

- Other specifications for voice markup;
- Standard multimedia notations (SMIL or a related standard);
- Standard visual markup languages e.g., HTML, CHTML, WML;
- Other relevant specifications, including ACSS;

The specification should avoid unnecessary differences with these markup languages." – W3C

Conformance in UIML:

Each of the markup Languages like VoiceXML, WML and HTML were designed taking into consideration the target platform's characteristics and capabilities. It is not possible to unify all of those languages. To be precise it is not possible to develop a common UI metaphor for those languages. Instead the UI widgets made available in each language should be used where appropriate. This is possible in UIML as the Interface developed can specify the UI widgets to be used in the mode (language) that is appropriate. Though a UI metaphor can be designed as a combination of generic widgets (least common denominator of UI widgets in each language) and language specific widgets (unique to each platform or language). Hence UIML encourages use of existing language's UI widgets where they cannot be unified. Also, in UIML synchronization cannot be specified more finely than the events. In case time stamping or finer synchronization is needed, UIML could be used with other synchronization markup languages like SMIL.

4.6.4.2 Identification of modalities

Requirement:

"The markup language will allow identification of the modalities available. This will allow an author to identify that a given modality is/is not present and as a result switch to a different dialog. E.g. there is a visible construct that an author can query. This can be used to provide for accessibility requirements and for environmental factors (e.g. noise). The availability of input and output modalities can be controlled by the user or by the system. The extent to which the functionality is retained when modalities are not available is the responsibility of the author.

The following is a list of use cases regarding a multimodal document that specifies speech and GUI input and output. The document could be designed such that:

1. When the speech input error count is high, the user can make equivalent selections via the GUI;
2. Where a user has a speech impairment, speech input can be deselected and the user controls the application via the GUI;
3. When the user cannot hear a verbal prompt due to a noisy environment (detected, for example, by no response), an equivalent prompt is displayed on the screen;
4. Where a user has a hearing impairment the speech output is deselected and equivalent prompts are displayed. " – W3C

Conformance in UIML:

In UIML all the available modes have to be specified. This might induce unnecessary redundancy. The author cannot query availability of modes. The equivalent functionality can be achieved in the Multimodal Interface Agent.

4.6.4.3 Transformable documents

4.6.4.3.1 Loosely coupled documents

Requirement:

"The mark-up language should support loosely coupled documents, where separate markup streams for each modality are synchronized at well-defined points. For example, separate voice and visual markup streams could be synchronized at the following points: visiting a form, following a link." – W3C

Conformance in UIML:

A mode can be associated to the entire document or interface just like it can be associated with individual parts. Modal integration could be done at different levels – application, document, dialog or individual widgets.

4.6.4.3.2 Tightly coupled documents

Requirement:

"The mark-up language should support tightly coupled documents. Tightly coupled documents have document elements for each interaction modality interspersed in the same document. i.e. a tightly coupled document contains sub-documents from different interaction modalities (e.g. HTML and voice markup) and has been authored to achieve explicit synchrony across the interaction streams.

Tightly coupled documents should be viewed as an optimization of the loosely coupled approach, and should be defined by describing a reversible transformation from a tightly coupled document to multiple loosely coupled documents. For example, a tightly coupled document that includes HTML and voice markup sub-documents should be transformable to a pair of documents, where one is HTML only and the other is voice markup only."

Conformance:

Tightly coupled documents in UIML are developed through the template mechanism. To transform tightly coupled documents into loosely coupled documents, the templates could be split into separate documents. For example,

```
<uiml>
    <template name="display" mode="HTML">
    <template name="voice" mode="Voice">
</uiml>
```

The above UIML file could be split into two different documents named display.uiml and voice.uiml.

4.6.4.4 Synchronization points

Requirement:

"The markup language should minimally enable synchronization across different modalities at well known interaction points in today's browsers, for example, entering and exiting specific interaction widgets:

- Entry to a form
- Entry to a menu
- Completion of a form
- Choosing of menu item (in a voice markup language) or link (HTML).
- Filling of a field within a form. " – W3C

Conformance in UIML:

Synchronization in UIML is only at the event level i.e. synchronization could be achieved at any part or widget level for which an event behavior is specified.

5 Conclusions

Below is the table of available platforms today and the corresponding API's used to develop user interfaces for these platforms.

Hardware	Operating System	Interface Application
Desktop computer	Windows (95/98/Me)	WinAPI
	Windows (NT/2000)	Java
	MacOS	MacAPI
		DHTML-Web browser
	Linux	Flash-Web browser
	Unix (BSD, Solaris, HP-UX, etc)	GnomeAPI
		KDE
		X-Windows
Mainframe	OS/60 OS-400	3270
	VMS	VT-100/220/240
	Unix	
Handheld computer	PalmOS	PalmAPI
	WinCE	WinCE-API
		DHTML
Smart Phone	WAP-forum	WML-WAP browser
		Do-Co-Mo
Standard Telephone	(none, server based)	DTMF –VoiceXML
		TAPI

Should a developer who would like his application made available to all the above platforms master the corresponding API's, when his application is meant to serve a single purpose? There is definitely a better solution to this problem and I believe it is embodied in UIML. UIML promotes component-based development. Much of the discussion in this thesis has shown that UIML is the ideal solution to build cross-platform interfaces and multimodal interfaces. Still a question arises whether UIML will succeed through the standardization process in spite of several advantages cited. If it succeeds, it's likely that the developers will adopt it for development. If it is not standardized, it is still a useful concept that could be put into use in the form of a data structure for maintaining and storing user interfaces for multiple platforms and multimodal systems. Software tools like Integrated Development Environment could use UIML format to store the interfaces developed in a compact way and retrieve individual interfaces when needed through rendering process. Also UIML files could be stored any where on the Internet and fetched using URL. There are plenty of ways UIML can be put into use whether it is standardized or not.

6 References

1. VoiceXML, VoiceXML Forum, http://www.voicexml.org/
2. Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, Jonathan E. Shuster, "UIML: An Appliance-Independent XML User Interface Language," 8th International World Wide Web Conference, Toronto, May 1999, http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html. Also appeared in Computer Networks, Vol. 31, 1999, pp. 1695-1708.
3. Marc Abrams, Constantinos Phanouri, "UIML: An XML Language for Building Device-Independent User Interfaces". http://www.harmonia.com/resources/papers/xml99Final.pdf
4. W3C Speech Interface Framework, http://www.w3.org/TR/voice-intro/
5. Speech Recognition Grammar Specification, http://www.w3.org/TR/speech-grammar/
6. Stochastic Language Models (N-Gram) Specification, http://www.w3.org/TR/ngram-spec/
7. Speech Synthesis Markup Language Specification, http://www.w3.org/TR/speech-synthesis
8. Specification of VoiceXML vocabulary for UIML, http://www.harmonia.com/products/voice/vocabulary.htm
9. Wireless Markup Language (WML), WapForum, http://www.wapforum.org/
10. Nuance, www.nuance.com/
11. Multimodal Requirements, http://www.w3.org/TR/multimodal-reqs
12. UIML1.0 specification, http://www.uiml.org/specs/UIML1/, 1997.
13. UIML2.0 specification, http://www.uiml.org/specs/UIML2/, 1999.
14. M.D. Wilson, D. Sedlock, J-L.Binot, P.Falzon, "An Architecture For Multimodal Dialogue".
15. Sharon Oviatt, Phil Cohen, Lizhong Wu, John Vergo, Lisbeth Duncan, Bernhard Suhm, Josh Bers, Thomas Holzman, Terry Winograd, James Landay, Jim Larson,

David Ferro, "Designing the User Interface for Multimodal Speech and Pen-based Gesture Applications: State-of-the-Art Systems and Future Research Directions".