# Simplifying Construction of Multi-Platform User Interfaces Using UIML

Mir Farooq Ali, Marc Abrams
Harmonia, Inc.
1715 Pratt Drive, Suite 3100,
Blacksburg, VA 24060, USA
farooq@harmonia.com, marc@harmonia.com

## Abstract

This paper presents one approach in simplifying the construction of multi-platform User Interfaces (UIs) using the User Interface Markup Language (UIML). A generic vocabulary is presented that includes a set of generic elements that can be used on any platform. In addition, a general process is presented for using the vocabulary to create generic UIML for various platforms.

## 1. Introduction

User Interface Markup Language (UIML) [3, 4] allows the construction of UIs for various platforms. One of the design goals of UIML is to "reduce the time to develop user interfaces for multiple device families" [2]. A related design rationale behind UIML is to "allow a family of interfaces to be created in which the common features are factored out" [1]. However, although UIML allows a multi-platform description of UIs, there is limited commonality in the platform-specific descriptions when platform-specific vocabularies are used. Some of the problems faced by UI developers in creating multi-platform UIs are discussed in this paper with the help of a sample UI created with UIML for three different platforms. One particular strategy to solve some of these problems is presented in the form of a *generic vocabulary*.

Some terminology that is used in the document is defined here.
***Application:*** The back-end logic behind a UI that implements the interaction supported by the UI is called an application.
***Device:*** A device is a physical object with which an end user interacts using a UI, such as a PC, a handheld or palm computer, a cell phone, an ordinary desktop voice telephone, or a pager.
***Vocabulary:*** This is the set of names, properties and associated events for UI elements available in any UI toolkit.
***UI Element:*** An UI element (widget) is the primitive building block provideD by any UI toolkit for creation of UIs.
***Platform:*** The combination of device, operating system and UI toolkit.
***UI toolkit:*** A toolkit is the markup language or software library upon which an application's UI runs.

The organization of the rest of the paper is as follows. Section 2 presents some problems associated with creating multi-platform UIs. Section 3 presents two vocabularies: a universal generic vocabulary and a generic vocabulary for GUIs. Section 4 presents a process to create a

multi-platform. The use of the generic vocabulary is demonstrated in this section using a sample interface.

## 2. Problems

Figure 1 shows the daily, weekly and monthly view of an Appointment Manager created for the Java™ platform. Figure 2 and 3 present the same for the Palm OS® and Wireless Markup Language (WML) platforms.

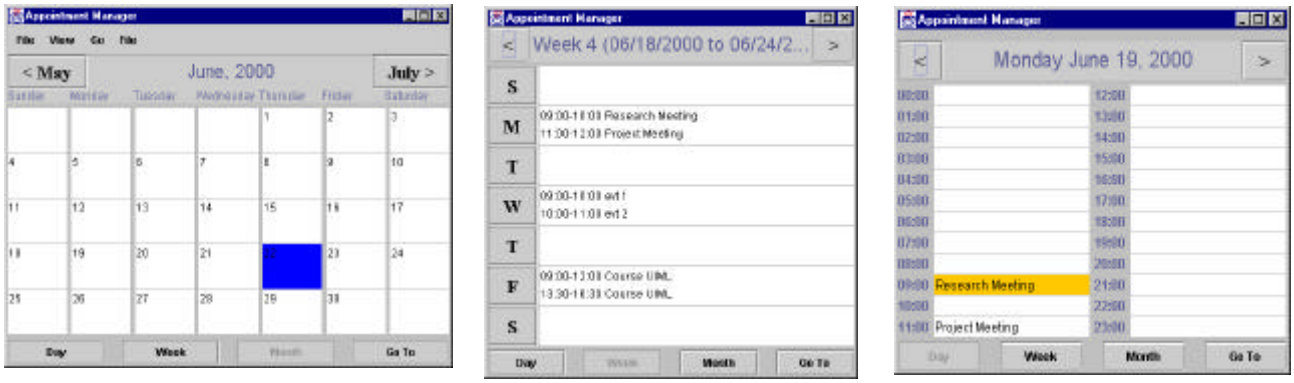


**Figure 1: Monthly, weekly and daily views for Java™**

All the three interfaces were created using UIML. However, Java™ Swing, Palm OS® and WML vocabularies were used in creating the UIs for each platform with separate renderers. Even though the interfaces have the same functionality, it was necessary to create separate UIML files for the interfaces. The actual number of lines that are common in the three files is very small due to differences in platform-specific interface elements and their properties. Some of the problems involved in creating these interfaces follow:

1. ***Separate vocabularies for different platforms:*** One of the rationales behind creating platform-specific vocabularies was that developers who are familiar with a particular platform should have all the features of the native platform available to them in UIML. This allows them to develop the same UIs using UIML that they would be able to build in the native platform. This also helps a developer who is familiar with a specific platform but new to UIML to transition to UIML quickly without a steep learning curve. However, because of the differences in the vocabularies for the three platforms, separate UIML code had to be written. Additionally, the three sets of vocabularies for the three platforms had to be mastered in order to create the UIML files.

2. ***Different layouts for different platforms:*** Java™ provides various layout managers for managing the physical layout of various elements on the screen. Palm OS® does not provide any layout manager of its own; instead one must give absolute positions for the various elements of the UI. WML also has limited layout management. Different approaches had to be taken to place the elements on the screen.

3. ***Reorganizing the interface to fit the available screen:*** This was a problem faced when trying to provide the full functionality of the Java™ and Palm OS® platforms on the smaller WML

platform. Since the screen size of a phone is tiny compared to the other platforms, the interfaces had to be carefully designed to provide all the features of the bigger platforms in the smaller platforms.


**Figure 2: Monthly, weekly and daily views for the Palm OS® platform**


**Figure 3: Monthly, weekly, daily and a consolidated view for WML**

The above are a few of the problems faced in creating multi-platform interfaces.

## 3. Generic Vocabulary

A *generic vocabulary* can be defined to be one vocabulary for all platforms. Creating a generic vocabulary can solve some of the problems outlined above. The vocabulary has two objectives: first, to be powerful enough to accommodate a family of devices, and second, to be generic enough to use without having to be an expert in all the various platforms. As a first step, a set of elements had to be selected from the platform-specific element sets. The elements selected in this step are available on at least two different platforms. After this several generic names had to be selected that represent various user interface elements in different platforms. Next properties had to be assigned for the generic elements.

However, even this approach has its pitfalls. It is definitely possible to create one generic vocabulary that accommodates every possible family of platforms. This has an undesirable side effect that the vocabulary becomes too bloated and contains a lot of UI elements that are not applicable for some family of devices. For example, there are a lot of UI elements that are specific to Graphical User Interfaces (GUIs). These elements might not be applicable to voice enabled platforms or devices.

Two vocabularies are developed to solve this problem. The first vocabulary is a truly generic vocabulary that can accommodate all possible families of devices and their UIs. This vocabulary has elements that can be assumed to be available on any device. A few of the elements might not

be available on some platforms with highly limited capabilities. A one-line pager is an example of such a device in which an interface element like a menu is not supported. The second set of vocabularies is specific to various families of devices. For example, there might be one vocabulary for GUIs and one for Voice interfaces. So a UI can be developed using elements from both the first and second vocabularies.

Table 1 presents the first generic vocabulary that includes the generic UI element name along with the corresponding platform-specific name for that element. The platforms considered for the vocabulary are Java™ [8], HTML 3.2 [9], Palm OS® [5], WML [6] and VoiceXML [7]. VoiceXML is the only non-graphical platform out of the above five platforms.

One point needs to be clarified in the table. When there are two or more elements separated by commas in the platform-specific column, it indicates that the generic element could be mapped to either of the elements in different circumstances.

It is evident from the table that there might be one-to-many mapping from a few of the generic elements to the platform-specific elements. Conversely, there might be some generic elements that do not have any corresponding element in a particular platform. Additionally, there might be a mapping in which one element is mapped to a combination of different elements. An example of this is the mapping of the GTopContainer element to `<html>` and `<body>` for HTML in Table 1. There are also cases in which the generic element is not mapped to any particular element in the platform-specific vocabulary even though a mapping exists for the element. This is dependent on the particular interface being developed. The above mapping descriptions are applicable for the properties of the elements too. The properties associated with the elements are not presented in this paper. For details about the elements and properties, please refer to the generic vocabulary specification [10].

**Table 1: Generic vocabulary**

| Generic User Interface Element | JAVA™ (Swing) | Palm OS® (Component) | HTML 3.2[1,2] (tag) | WML (tag) | VoiceXML (tag) |
|---|---|---|---|---|---|
| GButton | JButton | Command Button | <input type="button"> | <do> | <submit>, <clear> |
| GLabel | JLabel | Label |  | <p> | <prompt>, <block> |
| GIcon | Icon | Bitmap | <img> | <img> | <audio> |
| GSLTextRegion | JTextField | Field | <input type="text"> | <input> | <field> |
| GList | JList | Popup List | <ul>, <ol> | <select> | - |
| GDialog | JDialog | Alert Dialog, Process Dialog | - | - | - |
| GMenu | JMenu | Menu | - | | <menu> |
| GArea | JFrame, Window, JPanel, JScrollPane, JTabbedPane, JTable, JInternalFrame | Window | <form>, <table> | <card> | <form>, <menu> |
| GTopContainer | JFrame | Form | <html> + <body> | <wml> | <vxml> |

Table 2 displays a generic vocabulary for GUIs. VoiceXML is not considered for this vocabulary.

**Table 2: Generic Vocabulary for GUIs**

| Generic User Interface Element | JAVA™ (Swing) | Palm OS® (Component) | HTML 3.2 (tag) | WML (tag) |
|---|---|---|---|---|
| GMLTextRegion | JTextArea | Field | <textarea> | <input> |
| GCheckbox | JCheckBox | Checkbox | <input type="checkbox"> | - |
| GRadioButton | JRadioButton | Pushbutton | <input type="radio"> | - |
| GComboBoxList | JComboBox | Popup List | - | - |
| GScrollingList | JList | Popup List | - | - |
| GTree | JTree | - | - | - |
| GTable | JTable | Table | <table> | <table> |
| GMenuItem | JMenuItem | - | <select> | <choice> |
| GProgressIndicator | JProgressBar | Progress dialog | - | - |
| GSlider | JSlider | Slider | - | - |
| GToolBar | JToolBar | - | - | - |
| GToolTip | JToolTip | - | - | - |

[1] If scripting was used, then additional generic user interface elements could be used with HTML.
[2] Each HTML element in Tables 1 and 2, except for the top-level <html> and <body>, is enclosed within the  tag for positioning and presentation.

# 4. Demonstration of Generic Vocabulary

In this section the use of the generic vocabulary is demonstrated using an example interface. An example interface is presented followed by the process used in developing the generic UIML. The generic vocabulary can then be used to generate the platform-specific UIML for each of the three platforms.

The interface in this example is an interface for a weather system. It is loosely based on the weather forecasting web site at http://www.wunderground.com. Different renderings of the interface on three different platforms: Java™, Palm OS® and WML, are given in Figures 4, 5 and 6. Clearly, even though the content is similar, the organization of the interface on the three platforms is quite dissimilar. However, there are several common elements that can be factored out while creating generic UIML for the interface.

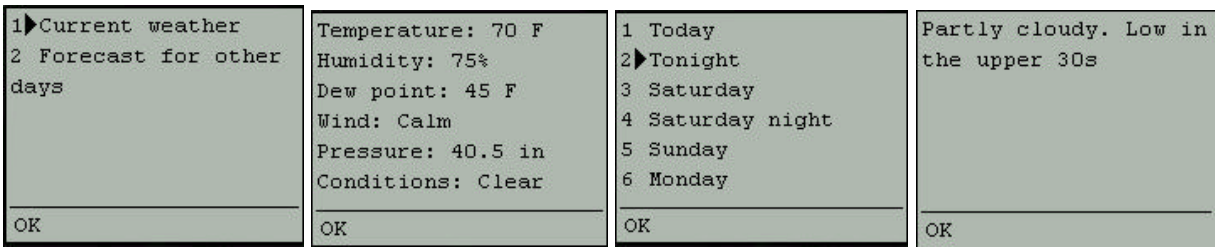


**Figure 4: WML version of Weather System**

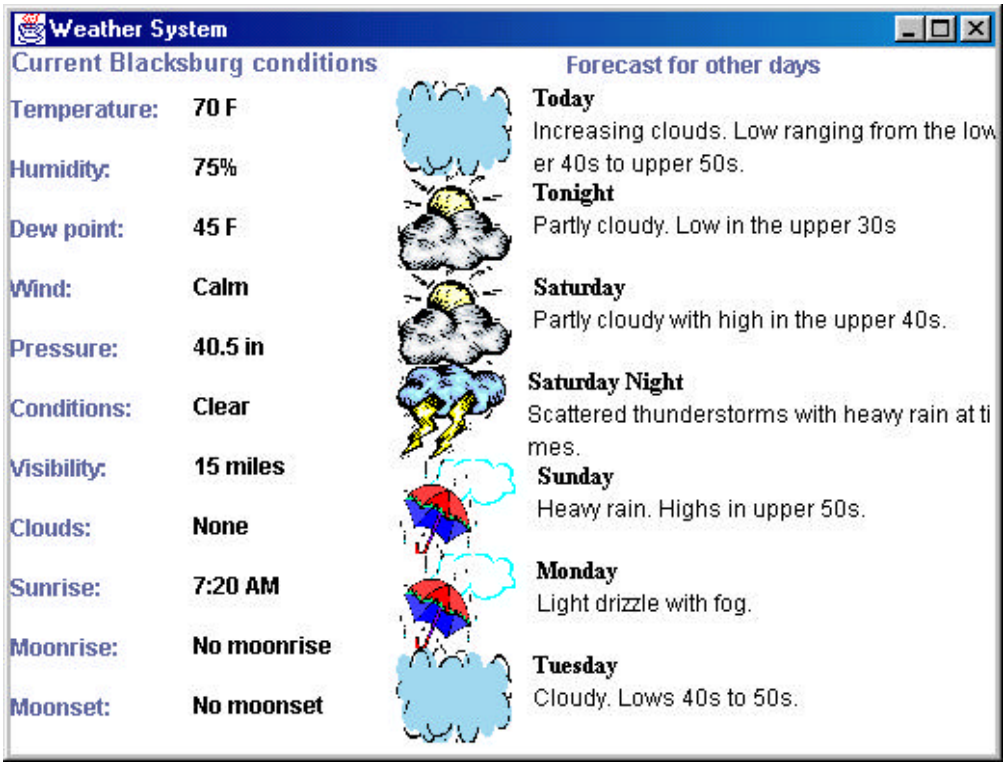


**Figure 5: Java™ version of weather system**

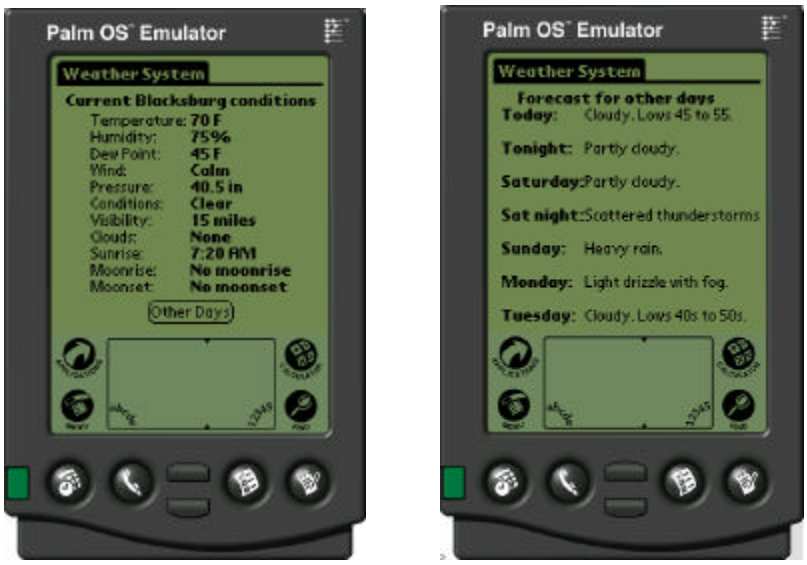


**Figure 6: Palm OS® version of weather system**

The high level organization of the information in the interfaces for the three platforms is shown in Figure 7.

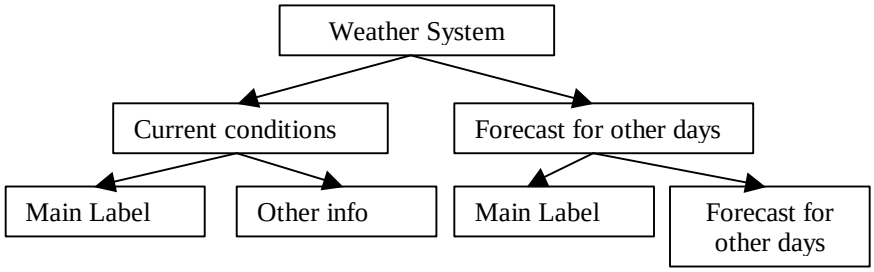


**Figure 7: High-level organization of interface**

This can be used to create the skeleton for the structure of the generic UIML file that is given below.

```
<structure>
    <part name="TopWeatherFrame" class="GTopContainer">
       <part name="CurrentConditions" class="GArea">
          <part name="CurrentLabel" class="GLabel"/>
          <part name="CurrentArea" class="GArea">
          ....
          </part>
       </part>
       <part name="Otherdays" class="GArea">
          <part name="ForecastLabel" class="GLabel"/>
          <part name="ForecastArea" class="GArea">
          ....
          </part>
       </part>
    </part>
</structure>
```

The CurrentPanel and ForecastPanel can then be expanded to include all the additional elements that are needed to create the interfaces. While creating the generic UIML, the safe approach is to look at all the different platforms and try to accommodate all possible elements that are present. Many of the elements in the generic UIML will not map to any element in the platform-specific vocabulary for any particular platform. For example, in Figure 6 for the WML interface, there is no bitmap or iconic representation for the various days. However, in the Java™ interface, we do have icons. So we should include the GIcon element in the generic UIML. This does not map to any element in the Palm OS® or WML platform.

The expanded UIML for the current example is given below.

```
<?xml version="1.0"  encoding="ISO-8859-1" ?>
<!DOCTYPE uiml PUBLIC "-//UIT//DTD UIML 2.0 Draft//EN"
    "UIML2_0d.dtd">

<uiml>

  <interface name="weatherinterface">

    <structure>
      <part name="TopWeatherFrame" class="GTopContainer">

        <part name="Current Conditions" class="GArea">
          <part name="CurrentLabel" class="GLabel"/>
           <part name="CurrentArea" class="Area">
                 <part name="Temp1Label" class="GLabel"/>
                 <part name="Temp2Label" class="GLabel"/>
                 <part name="Humid1Label" class="GLabel"/>
                 <part name="Humid2Label" class="GLabel"/>
                 <part name="Dewpoint1Label" class="GLabel"/>
                 <part name="Dewpoint2Label" class="GLabel"/>
                 <part name="Wind1Label" class="GLabel"/>
                 <part name="Wind2Label" class="GLabel"/>
                 <part name="Pressure1Label" class="GLabel"/>
                 <part name="Pressure2Label" class="GLabel"/>
                 <part name="Conditions1Label" class="GLabel"/>
                 <part name="Conditions2Label" class="GLabel"/>
                 <part name="Visibility1Label" class="GLabel"/>
                 <part name="Visibility2Label" class="GLabel"/>
                 <part name="Clouds1Label" class="GLabel"/>
                 <part name="Clouds2Label" class="GLabel"/>
                 <part name="Sunrise1Label" class="GLabel"/>
                 <part name="Sunrise2Label" class="GLabel"/>
                 <part name="Moonrise1Label" class="GLabel"/>
                 <part name="Moonrise2Label" class="GLabel"/>
                 <part name="Moonset1Label" class="GLabel"/>
                 <part name="Moonset2Label" class="GLabel"/>
           </part>
        </part>
        <part name="Otherdays" class="GArea">
          <part name="ForecastLabel" class="GLabel"/>
          <part name="ForecastArea" class="GArea">
           <part name="Today" class="GArea">
               <part name="TodayIcon"          class="GIcon"/>
```

```
                <part name="TodayLabelArea" class="GArea">
                  <part name="TodayLabel"         class="GLabel"/>
                  <part name="TodaydetailedLabel" class="GMLTextRegion"/>
                </part>
            </part>
            <part name="Tonight" class="GArea">
                <part name="TonightIcon"          class="GIcon"/>
                <part name="TonightLabelPanel" class="GArea">
                  <part name="TonightLabel"         class="GLabel"/>
                  <part name="TonightdetailedLabel" class="GMLTextRegion"/>
                </part>
            </part>
            <part name="Saturday" class="GArea">
                <part name="SaturdayIcon"          class="GIcon"/>
                <part name="SaturdayLabelPanel" class="GArea">
                  <part name="SaturdayLabel"         class="GLabel"/>
                  <part name="SaturdaydetailedLabel" class="GMLTextRegion"/>
                </part>
            </part>
            <part name="Satnight" class="GArea">
                <part name="SatnightIcon"          class="GIcon"/>
                <part name="SatnightLabelPanel" class="GArea">
                  <part name="SatnightLabel"         class="GLabel"/>
                  <part name="SatnightdetailedLabel" class="GMLTextRegion"/>
                </part>
            </part>
            <part name="Sunday" class="GArea">
                <part name="SundayIcon"          class="GIcon"/>
                <part name="SundayLabelPanel" class="GArea">
                  <part name="SundayLabel"         class="GLabel"/>
                  <part name="SundaydetailedLabel" class="GMLTextRegion"/>
                </part>
            </part>
            <part name="Monday" class="GArea">
                <part name="MondayIcon"          class="GIcon"/>
                <part name="MondayLabelPanel" class="GArea">
                  <part name="MondayLabel"         class="GLabel"/>
                  <part name="MondaydetailedLabel" class="GMLTextRegion"/>
                </part>
            </part>
            <part name="Tuesday" class="GArea">
                <part name="TuesdayIcon"          class="GIcon"/>
                <part name="TuesdayLabelPanel" class="GArea">
                  <part name="TuesdayLabel"         class="GLabel"/>
                  <part name="TuesdaydetailedLabel" class="GMLTextRegion"/>
                </part>
            </part>
          </part>
        </part>
      </structure>
  </interface>
</uiml>
```

Based on the above example, the following process should help in using the generic vocabulary to create the generic UIML:

1. Create a hierarchical model of the interface (e.g., Figure 7), partitioning the information into clearly identified groups. Try to group common elements as much as possible.
2. Enclose groups of elements within a container to allow for layout managers to be applied in case of platforms like Java™. These containers will map to null in case of smaller platforms like WML or Palm OS®. For example in the weather system, all the GLabels representing the current conditions have been grouped within a GArea.
3. Take the superset of all the elements for all platforms and accommodate them in the generic UIML. The use of GIcons in the above example illustrates this.

# 5. Conclusions

This paper presented a generic vocabulary for UIML that can be used to create multi-platform UIs. An example UIML file that used the generic vocabulary to build a sample interface was developed. Additionally, a process was described that aids the creation of the UIs.

The generic vocabulary presented in Tables 1 and 2 represents the authors' view of the mapping between the generic elements and platform-specific elements. This mapping has to be further refined. Currently, the mapping is primarily one-to-one or one-to many. There are many cases in which a single generic element maps to different combinations of elements on the same platform under different circumstances. A detailed explanation of such situations needs to be developed. The process outlined in Section 4 describes the process to use the generic vocabulary to create a UIML document. It does not describe what should be done when the target platform contains elements that have no corresponding generic element. Future work will describe what should be done in such cases.

A version of a UIML authoring tool exists in which a subset of the generic vocabulary mapping to three different platforms, Java™, HTML and WML has been implemented. An expanded set of the generic vocabulary will be implemented in the next version of the authoring tool.

# References


[1] Abrams, M., Phanouriou, C., Batongbacal, A., Williams, S., and Shuster, J., *"UIML: An Appliance-Independent XML User Interface Language"*, Proceedings of the World Wide Web Conference, Toronto, May 1999, http://www.harmonia.com/resources/www8_0599/index.htm

[2] Abrams, M. and Phanouriou, C., *"UIML: An XML Language for Building Device-Independent User Interfaces"*, XML' 99, Philadelphia, December 1999, http://www.harmonia.com/resources/xml99Final.pdf

[3] Phanouriou, C., ed. "*User Interface Markup Language (UIML) Draft Specification*", Language Version 2.0a, January, 2000, http://www.uiml.org/specs/docs/uiml20-17Jan00.pdf

[4] Phanouriou, C., "*UIML: An Appliance-Independent XML User Interface Language*", Ph.D. Dissertation, Virginia Polytechnic Institute and State University.

[5] Bey, C., Freeman, E., and Ostrem, J., *"Palm OS® SDK HTML Documentation"*, http://www.palmos.com/dev/tech/docs/palmos/

[6] Wireless Application Protocol Forum, "*Wireless Markup Language Specification*", http://www1.wapforum.org/tech/terms.asp?doc=WAP-191-WML-20000219-a.pdf

[7] VoiceXML Forum, "*VoiceXML Forum Version 1.0 Specification*", March 2000, http://www.voicexml.org/specs/VoiceXML-100.pdf

[8] Joy, B., Steele, G., Gosling, J., and Bracha, G., *"Java Language Specification"*, http://java.sun.com/docs/books/jls/index.html

[9] Ragget, D., "*HTML 3.2 Reference Specification*", World Wide Web Consortium, January 1997, http://www.w3.org/TR/REC-html32.

[10] Ali, M. F., ed. "*A Generic Vocabulary for UIML*", December 2000, to be posted on http://www.harmonia.com.