A Review of User Interface Description Languages for Mobile Applications

Nikola Mitrović*, Carlos Bobed*†, Eduardo Mena*†
*Dept. of Computer Science & Systems Engineering
University of Zaragoza, Spain
†Aragon Institute of Engineering Research (I3A), Spain
Email: mitrovic@prometeo.cps.unizar.es, {cbobed,emena}@unizar.es

***Abstract*—Adapting a graphical user interface (GUI) for various user devices is still one of the most interesting topics in today's mobile computation. The benefits of separating the specification of the GUI from its implementation are broadly accepted. However, it is not clear which are the benefits and disadvantages of current GUI specification languages in order to define and develop multiplatform mobile applications which can adapt dynamically to different devices with different features. In this paper, we review User Interface Description Languages (UIDLs) that can be used currently to specify GUIs in mainstream mobile platforms. We analyze their features and usefulness for dynamic adaptation of GUIs to heterogeneous mobile devices. All reviewed UIDLs are suitable for developing mobile applications; however, there are no UIDLs that will truly be able to operate across multiple platforms.**

***Index Terms*—Adaptive GUI; Mobile Computing.**

I. Introduction

The use of mobile devices and applications is increasing. Adapting GUIs to different mobile devices is still one of the most interesting problems in mobile computing as modern devices vary considerably in their properties (e.g., screen size, resolution, user input controls). The benefits of working with User Interface Definition Languages (UIDLs) [1] to specify GUI is that such a specification can be re-used and adapted to different devices automatically. This approach has been accepted by the GUI researchers a long time ago [2], and it has been applied to multiple devices in the Personal Digital Assistant (PDA) era [3]. However, there does not exist yet a standard UIDL that is widely used by software developers, let alone by mobile apps developers.

In this paper, we review main User Interface Description Languages (UIDLs) that can be used currently to specify GUIs in mainstream mobile platforms. While there is a large number of research-based UIDLs (such as UIML [2], UsiXML [4], or Maria XML [5], to name a few), these UIDLs have limited support and implementation code outside their respective research institutions, and, thus, we will focus on the *mainstream* UIDLs that have strong adoption in at least one of the significant technology ecosystems.

Previous UIDL reviews [6][7][8] focused on theoretical UIDLs, devices and platforms of the time, and their usefulness for Human-Computer Interaction adaptation in general, as opposed for mobile application suitability. These prior reviews did not analyze modern, industry accepted, UIDLs such as Android XML [9] or XAML [10], and did not consider today's mainstream mobile devices and their market uptake. We analyze the features of mainstream UIDL languages, and evaluate them from the point of view of being used by mobile applications to allow a dynamic adaptation of such specifications to heterogeneous modern mobile devices. We focus on their ease of use, and the availability of visual tools, among other parameters. We consider the following two categories of UIDLs, based on their relationship with the mobile platforms and Web browsers:

1) *UIDLs specific for mobile devices.* These UIDLs are specifically developed for a particular mobile platform, e.g., Android or iOS devices.
2) *UIDLs associated with Web browsers.* These UIDLs are linked to one or more Web browsers, and can be used for mobile application development.

Finally, for the purpose of this review, we use as an example a simple currency converter, which converts numeric amounts between three currencies. This example application was developed for each UIDL that is reviewed in this paper in order to evaluate the usefulness of the provided tools, and its applicability to multi-device and multi-platform use. Full specifications of the example application in different UIDLs can be found in [11].

The remainder of this paper is structured as follows. Section II presents the UIDLs used in popular mobile platforms. In Section III, we review UIDLs that are associated with Web browsers. In Section IV, we analyze and compare the above approaches. Finally, Section V gives some conclusions and future work.

II. UIDLs Specific for Mobile Devices

In this section, we review the UIDLs used in the market leading mobile platforms, namely, Android XML (Android) and Storyboards (iOS); both platforms together reach a 92% of market share [12]. We also include other relevant UIDLs for mobile devices such as XAML (Windows 10), and QML (Ubuntu OS).

A. Android XML

Android is the most widely used operating systems when it comes to mobile devices (i.e., smartphones and tablets): An-

droid market share is estimated at 60.99%. It is a Linux-based operating system whose middleware, libraries, and APIs are written in C. Android supports Java code as it uses a Java-like virtual machine called *Dalvik* (substituted by ART from Android 5.x onward). In Fig. 1, we show an excerpt of the Android XML code of our example application and a screenshot rendered in an Android N emulator.

```
<LinearLayout
  ...
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content"
      android:text="Currency Converter"
            android:id="@+id/textView" />
  <LinearLayout android:orientation="horizontal" ... >
    <TextView ... android:id="@+id/textView2"
            android:text="Quantity:" />
    <EditText ... android:id="@+id/Qty" .../>
  </LinearLayout>
  <LinearLayout android:orientation="horizontal" ... >
    <TextView ... android:id="@+id/textView3"
            android:text="From:" />
    <RadioGroup ...  android:id="@+id/From">
      <RadioButton ... android:id="@+id/eurFrom"
                android:text="Euros" ... />
      <RadioButton ... android:id="@+id/usdFrom"
                android:text="US Dollars" ... />
      <RadioButton ... android:id="@+id/gbpFrom"
                android:text="British Pounds" ... />
    </RadioGroup>
    <TextView ...
            android:id="@+id/textView4"
                android:text="To:" />
        ...
  </LinearLayout>
  <LinearLayout android:orientation="horizontal" ...
    <TextView ... android:id="@+id/textView5"
            android:text="Result:" />
    <TextView ... android:id="@+id/output" />
  </LinearLayout>
  <Button ... android:id="@+id/convert"
    android:text="Convert" />
</LinearLayout>
```

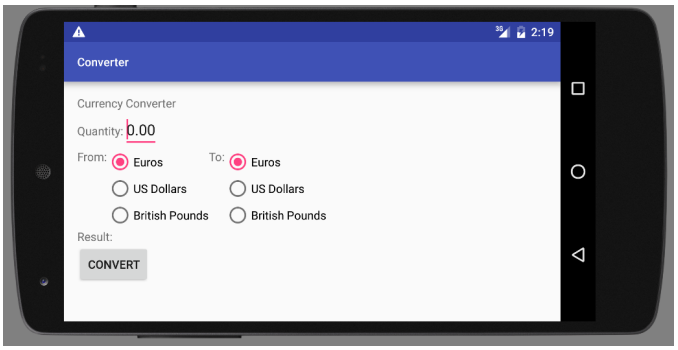

Fig. 1. Excerpt of the Android XML specification and rendering for the sample app.

Android applications are built using *Activities*, which can be regarded as windows in a usual desktop environment, and *Services*, which are background processes without GUI. Graphical User Interfaces are defined using a XML-based notation named Android XML [13][9].

Given the heterogeneity of Android devices, support for GUI adaptation to different devices is provided using layout elements and visual behavior policies. Android also provides further mechanisms to deal with different device display capabilities and adapt the interface, but it is the developer who is in charge of developing the application in a responsive and plastic way [14].

The developer can specify behavioral aspects in Android XML in different ways, such as event handling. Moreover, within the Android XML document, the application developer can state implicit navigation via the definition of *Intents*, objects which represent the intention of the application, and allows development of applications in a service oriented way. This service oriented model allows applications to use activities from other applications using a built-in broker mechanism.

GUI and UIDL development can be performed using the Android Studio SDK's visual editor. Android XML is sufficiently developer-friendly and can be edited manually when required, e.g., to refine GUI behavior.

B. Storyboards (iOS)

Similarly to Android, Apple's iOS also adopts a XML-driven interface via *Storyboards* [15]. Apple's approach goes further than Android: Storyboards use *views* (similar to Android's *activities*), but they can also explicitly include navigation aspects of user interaction in the GUI specification. Thus, a storyboard provides a comprehensive view of the whole application and the workflow of interactions.

The set of elements provided by iOS to define UIs does not contain some common visual elements such as radio and check button widgets (although the platform allows custom widgets to be developed if required). In Fig. 2, we present a screenshot of our example application rendered in iOS emulator; note that choices are made using switches, not radio or check buttons. We have not included the iOS storyboard code due to space limitation (the full code is available in [11]).

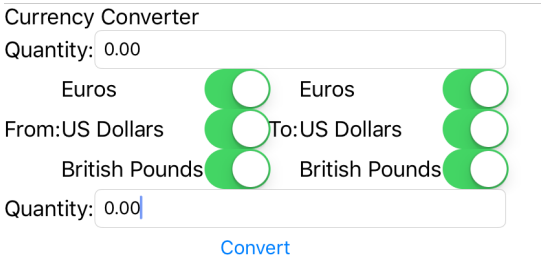

Fig. 2. Sample app as rendered in iOS emulator from its storyboard specification.

The model adopted by iOS devices is more closed than the one adopted by Android. When developing applications for iOS, the developer has lower heterogeneity of devices than Android, but the interface guidelines are more strictly dictated by the environment. The definition of the application appearance is done via a visual editor integrated in Apple's Xcode environment. While in Android (and other used UIDLs) the interface definition can be easily defined (or at least edited) by the developer without visual help, storyboard XML language is clearly designed to be machine-generated and not

edited manually. The application logic in iOS can be developed using Objective-C or Swift programming languages.

C. XAML: eXtensible Application Markup Language

XAML [10] is the UIDL developed by Microsoft and has been used in several of their technologies (e.g., .NET 4.0, Silverlight). XAML is also used to specify GUIs for the Windows 10 platform. This makes XAML available not only on fixed Windows computers but also on mobile devices. See Fig. 3 for an excerpt of the XAML code of our example application developed as an UWP application, and its rendering on a Windows 10 Desktop.

```
<Page x:Class="Converter.Converter" ... ">
   <StackPanel >
      <StackPanel >
         <TextBlock> Currency Converter</TextBlock>
         <StackPanel Orientation="Horizontal">
            <TextBlock Text="Quantity:"/>
            <TextBox Text="0.00" x:Name="Qty"/>
         </StackPanel>
         <StackPanel Orientation="Horizontal">
            <TextBlock Text="From:"/>
            <StackPanel>
               <RadioButton x:Name="EurFrom"
                  GroupName="From">Euros</RadioButton>
               <RadioButton x:Name="UsdFrom"
                  GroupName="From">US Dollars</RadioButton>
               <RadioButton x:Name="GbpFrom"
                  GroupName="From">British Pounds
               </RadioButton>
            </StackPanel>
            ...
         </StackPanel>
      </StackPanel>
      <StackPanel Orientation="Horizontal">
         <TextBlock Text="Result:"/>
         <TextBlock x:Name="Output" Text=""/>
      </StackPanel>
      <Button x:Name="Convert">Convert</Button>
   </StackPanel>
 </Page>
```

Fig. 3. Example of the XAML specification and rendering for the sample app.

Microsoft made an important effort to unify development of applications for different Microsoft platforms under the Universal Windows Platform (UWP) programme [10]. UWP applications share a basic API and GUI elements which are then extended and specialized for specific device families. As long as the developer restricts its application to the use of the basic API and XAML elements the application will run on all devices that are compatible with UWP.

Although XAML is used only in Windows-based devices, there is still a need to adapt GUIs to different devices. This adaptation is quite similar to Android: developers need to provide layout elements and policies in order to adapt (to a certain extent) automatically the GUI to the specific device.

D. QML

QML is an UIDL associated to Qtgraphical libraries [16] and it has been adopted as UIDL for Ubuntu OS applications. QML is a JSON-like language, where graphical elements are grouped in libraries which can be imported as needed (thus providing an extension mechanism). Fig. 4 shows an QML code excerpt for our example application and a screenshot of the GUI as rendered by Qt Designer (Ubuntu Mate Desktop).

```
import QtQuick 2.1 ...
ApplicationWindow {
   id: applicationWindow1
   title: qsTr("Converter")
   ColumnLayout {
      id: columnLayout1
      anchors.rightMargin: 0
      anchors.bottomMargin: 0
        ...
        Label {
         id: label1
         ...
         text: qsTr("Currency Converter")
      }
      RowLayout {  /* From Radio button */
         id: rowLayout2
         ...
         Label {
            id: label3
            ...
            text: qsTr("From:") }
         ColumnLayout {    /* inside RowLayout */
            id: columnLayout3
            ...
            ExclusiveGroup {id:from}
            RadioButton {
               id: eurFrom
               ...
               text: qsTr("Euros")
               checked: true
               exclusiveGroup: from }
            RadioButton {
               id: usdFrom
               ...
               text: qsTr("US Dollars")
               exclusiveGroup: from }
            RadioButton {
               id: gbpFrom
               ...
               text: qsTr("British Pounds")
               exclusiveGroup: from}
                 }...}...}...}
```

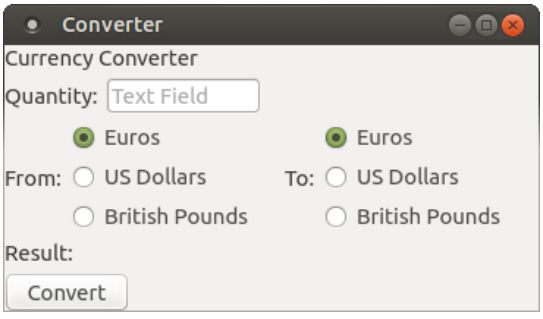

Fig. 4. Example of the QML specification and rendering for the sample app.

Being a general purpose UIDL language, QML (since Qt 5.1) also provides layout mechanisms in order to support

device adaptation. Previously, Qt Quick support for windows resizing (not device adaptation) was limited to the use of *positioners* for items, and *anchors* to layout children GUI elements. Qt provides bindings to multiple programming languages and platforms, which makes the adoption of QML a feasible solution for multiplatform development. Moreover, as with Android XML and XAML, the specification is developer-friendly and can be edited using visual tools or manually.

III. UIDLs Associated with Web Browsers

Apart from developing ad hoc apps for each of the mobile platforms, the development of applications using Web technologies has increased as Web browsers are broadly available for fixed and mobile devices. These applications use the Web browser (or its engine) as a kind of runtime middleware, where an application can be deployed independently of the underlying mobile platform (with some limitations). This section reviews two UIDLs that are used by Web browsers: HTML5, the most widely used UIDL as it is supported by both mobile and desktop computers, and eXtensible User Interface Definition Language (XUL), used by the Mozilla Foundation.

A. HTML5

HTML5 [17] is a the new version of HyperText Markup Language, the language for structuring and presenting content on the Web. While HTML5 can be considered to be mainly content-oriented, it offers several form tags to interact with the user, which makes it also suitable to define user interfaces. For our review, we are considering plain HTML5, without any JavaScript library extensions.

Fig. 5 shows an excerpt of the HTML5 specification for the sample app and a screenshot of its rendering in Firefox. Note that plain-HTML tag `<table role="presentation">` was used to specify the layout of the GUI. The use of this tag has been discouraged by the W3C HTML5 Recommendation Document and use of CSS [17] is advised instead. This requires GUI developers have to manage both HTML5 and CSS descriptions to specify the required GUI layout.

HTML5 introduces new tags to include different types of content that are now directly supported by browsers (e.g., `<video>`, `<audio>`, `<canvas>`, ...). Several new control forms [17] are introduced too (e.g., date, color, search, etc.). Moreover, some tags have been included to define a basic web document layout (e.g., `<header>`, `<nav>`, `<footer>`, ...). However, responsiveness and layout adaption is delegated to CSS (usually combined with JavaScript).

The technology stack HTML5+CSS+JavaScript has gained momentum in mobile applications thanks to: 1) ubiquity of Web browsers, 2) the usefulness of client-server model to allow frequent content updates (rather than providing application updates), and 3) the introduction of cross-platform HTML5 code engines, such as Apache Cordova [18] or Crosswalk [19]. Firefox OS [20] and Ubuntu OS applications can also be implemented using this technology stack. While multiplatform applications can be developed using HTML5 and Cordova

```
<html> ... <body>
  <table role="presentation">
    <tr> <td> Currency Converter </td> </tr>
    <tr> <td>
      <table>
        <tr> <td>
          <table role="presentation">
            <tr> <td> Quantity </td>
              <td>
                <input type="text"
                  id="Qty" value="0.00"/> </td> </tr>
          </table> </td> </tr>
        <tr> <td>
          <table role="presentation">
            <tr> <td> From: </td>
              <td>
                <input type="radio" name="From"
                  value="eurFrom"> Euros <br>
                <input type="radio" name="From"
                  value="usdFrom"> US Dollars <br>
                <input type="radio" name="From"
                  value="gbpFrom"> British Pounds </td>
              ...
          </table> </td> </tr>
        <tr> <td>
          <table role="presentation">
            <tr> <td> Result: </td>
              <td id="output"> </td>  </tr>
          </table> </td> </tr>
        <tr> <td>
          <table role="presentation">
            <tr> <td>
                <input type="button" name="Convert"
                  value="Convert">
              </td> </tr>
          </table> ...
    ...
</body> </html>
```

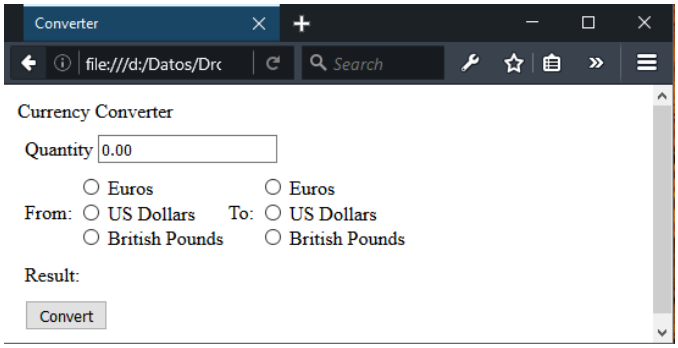

Fig. 5. Example of the HTML5 specification and rendering for the sample app.

or Crosswalk, such applications need to be installed on each computer in the same way as regular applications (to run, some of them have to be bundled along with a particular runtime).

B. XUL: eXtensible User interface definition Language

XUL [21] is the UIDL developed and supported by Mozilla in its Gecko engine. XUL allows development of multiplatform interfaces by providing a GUI specification that is very abstract and not related to any specific devices or platforms. In Fig. 6, we present an excerpt of the XUL specification of our sample app and its rendering in Firefox.

In order to provide adaption to the different capabilities of the devices, XUL relies both on predefined layouts, and customization via CSS and JavaScript. While it is mainly oriented to window-based GUIs, the widgets and basic layouts

provide developers with a higher level abstraction than other similar languages (e.g., HTML5).

```
...
<window title="Converter" xmlns=" ... /there.is.only.xul">
<vbox>
   <label control="lblAll" value="Currency Converter"/>
   <hbox>
      <label control="lblQty" value="Quantity:"/>
      <textbox value="0.00" id="Qty"/>
   </hbox>
   <hbox>
      <label control="lblFrom" value="From: "/>
      <radiogroup orient="vertical" id="From" ...>
         <radio id="EurFrom" label="Euros"/>
         <radio id="UsdFrom" label="US Dollars"/>
         <radio id="GbpFrom" label="British Pounds"/>
      </radiogroup>
      ...
   </hbox>
   <hbox>
      <label control="lblOutput" value="Result: "/>
      <label id="Output" control="Output" value=""/>
   </hbox>
   <hbox>
      <button id="Cnv" label="Convert" oncommand="convert()"/>
   </hbox>
</vbox>
</window>
```

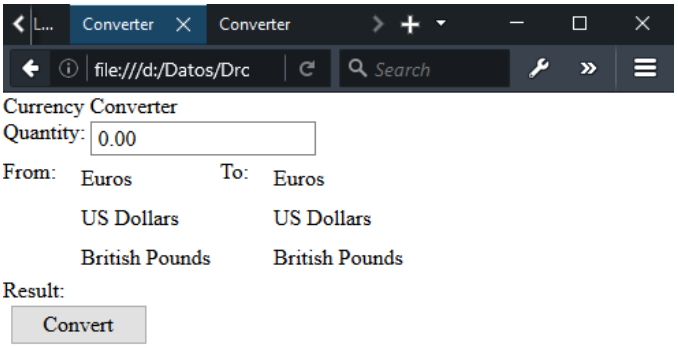

Fig. 6. Example of the XUL specification and rendering for the sample app.

XUL has been adopted as the UIDL for developing extensions for Firefox Web browser, but this use might be discontinued in favor of adopting WebExtensions (mainly due to security reasons). In fact, Firefox OS apps are developed using the full Web technology stack (HTML5, CSS, and JavaScript). The application logic for XUL applications using Gecko engine is mainly developed using JavaScript.

IV. Comparison

After defining our sample Currency Converter application in all the reviewed UIDLs to grasp the philosophy of each language, we analyze the pros and cons of each of them from the point of view of using them to develop the GUI of applications that must run on very different fixed or mobile computers. Thus, in Table I, we can see different appealing characteristics related to multiplatform development, namely:

- Multi-device: It refers to whether the UIDL takes into consideration devices with different characteristics (e.g., screen size).
- Multi-OS: Considers if the UIDL can be used outside the boundaries of a particular OS (or mobile platform).
- Multi-language: It refers to whether the UIDL can be used with different programming languages to develop the application logic.
- Visual Editor: Considers existence of tools that allow developers to design UIDL-based GUIs in a visual manner, abstracting developers from UIDL's syntax.
- Friendly Markup: Considers, regardless the existence of a visual editor, whether the UIDL specification can be edited manually by the developer easily or, alternatively, is the UIDL too difficult to edit manually due to e.g., complexity and number of UIDL elements.
- Layout Support: It refers to whether the UIDL allows developers to select predefined GUI layouts.

Regarding multi-device support, all the analyzed languages can be used in a multi-device target environment. However, it has to be noted that both HTML5 and XUL require a Web browser (or an engine) to be available for each device. Regarding operating systems support for UIDLs, Android XML and Storyboard are tightly bounded with Android and iOS, respectively. Moreover, note that XAML, while is suitable to develop GUIs for several platforms, is restricted to work within Windows devices.

Concerning supported programming languages, the logic of the applications developed with Android XML is, at first, restricted to Java. In iOS (Storyboards), the developer can use Objective-C or Swift. XAML can be used by different programming languages that are available under the .Net platform (e.g., Java, C#, etc.). For QML, JavaScript is recommended for developing apps in Ubuntu OS, but many other languages can be used when used alongside Qt libraries (if appropriate language bindings are available). The application logic of applications in HTML5 needs to be developed in JavaScript. Last but not least, there are some efforts to use XUL with different languages (such as Java) but they seem discontinued.

The existence of a friendly visual editor is not a problem for any analyzed language but for XUL. However, XUL has a really friendly markup which can be easily used to define the UIs. On the other side, in practice, iOS Storyboards require a visual editor due to the verbosity of its GUI specifications.

Finally, all languages but HTML5 support layout elements (and mechanisms) that help the developer to describe the interfaces in an adaptive way, which is a very important feature for multidevice application development. The case of HTML5 is special as layout is delegated almost completely to the use of complementary CSS (there are tags that are used to serve as entry points for this, i.e., `<div>`).

V. Conclusions and Future Work

In this paper, we have reviewed several popular User Interface Definition Languages (UIDLs) from the point of view of their use in the context of mobile applications. The objective of our evaluation was to understand usefulness, benefits, and drawbacks of UIDLs when adapting to multiple mobile devices or different user contexts. We defined an example application in each UIDL to help the evaluation.

TABLE I
COMPARISON OF THE REVIEWED UIDLS.

Feature	Android XML	Storyboard	XAML	QML	HTML5	XUL
Multi-device	✓	✓	✓	✓	✓[2]	✓[2]
Multi-OS	×	×	✓[1]	✓	✓[2]	✓[2]
Multi-language	×	✓	✓	✓[3]	×	✓
Visual Editor	✓	✓	✓	✓	✓	×
Friendly Markup	✓	×	✓	✓	✓	✓
Layout Support	✓	✓	✓	✓	×	✓

[1] Provided that all devices are Windows-based.
[2] Provided a suitable engine (e.g., Web browser or app engine) is available.
[3] Provided there is a binding of Qt libraries to such a language.

As summary, all reviewed UIDLs can be used in mobile applications and adapted to multiple devices. However, we found that the most popular languages have strong dependency on their underlying platform and vendor (Android XML, XAML, Storyboards). These UIDLs are used only on Android, Apple, or Microsoft devices and do not have any support on other platforms. QML, whilst being less vendor-specific, requires language and platform bindings which may not be available or are difficult to use. On the other side, HTML5 is widely accepted as UIDL but requires several technologies to be combined in order to deliver GUIs (i.e., CSS, JavaScript, Web Browser). This makes the development of applications more complex and potentially less portable between different device-platform combinations. Mozilla's XUL, on the other hand, appears to be combining the features of other UIDLs and allows applications to be developed outside the Mozilla platform (i.e., without having a Web Browser).

We have shown that currently there does not exist a good multi-OS oriented UIDL which can be broadly adopted for developing applications whose GUI is correctly presented on very different mobile devices. Whenever an application needs to be delivered on multiple platform-device combinations, the user interface is not likely to be reused unless in vendor-specific situations (e.g., XAML for Microsoft environments, or Android XML for Android). HTML5, as stated before, requires several technologies to deliver a functional GUI description. Finally, XUL may offer the best chance of redeploying the user interface given that there are some implementations outside Mozilla's Web browser engine, albeit some of these appear to be discontinued.

As future work we plan to test different UIDLs by implementing more complex applications in multiple devices and user contexts. Furthermore, we will analyse performance implications of choosing a specific UIDLs, and their usefulness for advanced GUIs.

ACKNOWLEDGMENT

This work was supported by the CICYT project TIN2013-46238-C4-4-R and DGA-FSE.

REFERENCES

[1] N. Mitrović, E. Mena, and J. A. Royo, *Chapter XIX - Adaptive Interfaces in Mobile Environments: An Approach Based on Mobile Agents*. Information Science Reference, 2007, pp. 302–317.

[2] M. Abrams, C. Phanouriou, A. L. Batongbacal, S. M. Williams, and J. E. Shuster, "UIML: an appliance-independent XML user interface language," *Computer Networks*, vol. 31, no. 1116, pp. 1695–1708, 1999.

[3] N. Mitrović and E. Mena, "Adaptive user interface for mobile devices," in *Proceedings of the 9th International Workshop on Interactive Systems. Design, Specification, and Verification (DSV-IS'02)*, vol. 2545. Springer LNCS, June 2002, pp. 47–61.

[4] J. Gonzalez-Calleros, J.-P. Osterloh, R. Feil, and A. Ldtke, "Automated UI evaluation based on a cognitive architecture and UsiXML," *Science of Computer Programming*, vol. 86, pp. 43–57, 2014.

[5] F. Paterno, C. Santoro, and D. S. Lucio, "MARIA: a universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments," *ACM Transactions on Computer-Human Interaction*, vol. 16, no. 4, pp. 219–224, 2009.

[6] J. Guerrero-Garcia, J. M. Gonzalez-Calleros, J. Vanderdonckt, and J. Muoz-Arteaga, "A theoretical survey of user interface description languages: Preliminary results," *Latin American Web Congress*, pp. 36–43, 2009.

[7] N. Souchon and J. Vanderdonckt, "A review of xml-compliant user interface description languages," in *Proceedings of the 10th International Workshop on Interactive Systems. Design, Specification, and Verification (DSV-IS'03)*, vol. 2844. Springer LNCS, 2003, pp. 377–391.

[8] J. Engel, C. Herdin, and M. Christian, "Review of user interface definition languages," in *Proceedings of the 6th Forum Medientechnik*. VWH, 2014, pp. 183–198.

[9] J. Morris, *Android User Interface Development*. Packt Publishing, 2011.

[10] A. Nathan, *Building Windows 10 Applications with XAML and C# Unleashed (2nd Edition)*. Sams, 2016.

[11] N. Mitrovic, C. Bobed, and E. Mena, ADUS: Full UIDL code and samples at http://sid.cps.unizar.es/projects/ADUS/UIDLs, last accessed 30th Aug 2016.

[12] MarketShare, http://www.netmarketshare.com/, last accessed 30th Aug 2016.

[13] R. Rogers, J. Lombardo, and M. Blake, *Android Application Development*. O'Reilly, 2009.

[14] A. Demeure, G. Calvary, J. Coutaz, and J. Vanderdonckt, "The comets inspector: Towards run time plasticity control based on semantic network," in *Proceedings of 5th International Workshop on Task Models and Diagrams for UI Design (TAMODIA'06)*, vol. 4385. Springer LNCS, October 2006, pp. 324–338.

[15] M. Neuburg, *Progamming iOS 9*. O'Reilly, 2015.

[16] R. Rischpater, *Application Development with Qt Creator (2nd Edition)*. Packt Publishing, 2014.

[17] W3C, HTML5 W3C Recommendation, http://www.w3.org/TR/html/, last accessed 30th Aug 2016.

[18] J. M. Wargo, *Apache Cordova 4 Programming*. Addison Wesley, 2015.

[19] Crosswalk-Project, http://www.crosswalk-project.org, last accessed 30th Aug 2016.

[20] T. Pant, *Learning Firefox OS Application Development*. Packt Publishing, 2015.

[21] V. Bullard, K. T. Smith, and M. C. Daconta, *Essential XUL Programming*. Wiley, 2001.