

MULTIPLE DEVICE MARKUP LANGUAGE A RULE APPROACH

**JINESH PAREKH, PAUL JOHNSON
DEPAUL UNIVERSITY
CHICAGO, IL**

Email: rajparekh60@hotmail.com , pdj@dls.net

Abstract

It is not uncommon to have several GUI representations for a single application, thereby increasing the versatility of the application. This requires developing separate graphical user interface for each platform. Future extensions of the application to other platforms would again involve developing the GUI representations for those platforms. This is of course redundant and time consuming.

The purpose of this paper is to propose a rule-based approach to automate the process of GUI generation based on a single, XML based GUI definition. An attempt is made to address the issue of mapping the functionalities of an application to vastly different capabilities and constraints of the GUI platforms.

Introduction

To revert to the growing demand of accessing the data ubiquitously, today's Internet instruments features various approaches of representing data. These approaches involve using touch screens, styli, handwriting and voice recognition, speech synthesis, tiny screens, and more [2]. Each of these approaches uses different development languages. For example, Java Swing, XUL [3] or HTML is used for desktop applications; WML or MIDP is used for mobile applications where as voice XML, JSML or Speech ML is used for voice-enabled devices [2]. Thus writing applications that could be supported by these instruments requires developing and marinating the code for each of them separately. This demand's for expertise in largely different languages for a developing and marinating a single application. The increase in the use of XML [1] foresees the development and popularity of many other new languages. So to later support the application on one of these future languages would again demand expertise in that language for developing and maintaining the application. This makes the development process expensive and time consuming. The problem demands for a universal approach that would allow defining the graphical user interface only once and generate code for mapping it to various devices and their features automatically.

With the advent of these instruments and various methods of representing and manipulating data comes the urge for ubiquitous control [4]. This means that the users want to control the appliances in their environment with their mobile devices or desktop computers. For example, the user wants to control appliances such as microwave oven, car stereo, fax machine, music system, copier, CD player and more. These appliances are

functionally different from the each other. The problem here is different in the sense that in this case the GUI has to be generated on the fly on the controlling device.

However in both of the above cases the problem boils down to generating the graphical user interface automatically. The goal of this project is to device a layer of abstraction for defining the GUI representation of the application which when fed to the underlying framework would automatically generate the code for the required instrument. This approach also fosters the separation of the graphical user interface from the structure and the behavior of the business logic.

This paper will first describe some of the background work and the motivation for it. Later it will define a rule-based approach as a solution to the above-mentioned problem and describe the architecture of the framework. Finally it will conclude with a discussion possible future work.

2. Background

This section talks about some of the related research in the area and attempts to compare them with the approach that this paper proposes. Interestingly, all of these projects have one thing in common - they all are XML based approaches.

2.1 XML user interface language (XUL)

The motivation behind developing XUL(“zool”) was to build a cross platform application for developing graphical user interfaces. It is an XML based system that aids in defining the GUI rather than developing. It is platform independent and can run on virtually any operating system that runs Mozilla browser on it. Also it is a write once and run anywhere application. Thus all the Mozilla’s core applications are developed using a single code base. It also provides a layer of separation between the GUI and the business logic of the application [5].

XUL features a concept of having a single development environment for all the applications. The motivation of XUL was to support a cross platform application development and was not targeted for web development. However it does require, Mozilla running on the system and will not be able to work otherwise.

2.2 Multimodal Interactive User Interfaces for Mobile Multi-Device Environments

This research targets a very specific genre of devices – mobile devices. The motivation here is to address the problem of mapping the application functionalities to different constraints and capabilities of mobile devices. The claim of the project is that it is not possible to develop a single graphical user interface representation for all the mobile devices as they all have different constraints, screen size and display capabilities [6]. This project is very similar to the project that this paper describes but it is builds on a more specific domain-mobile devices.

2.3 Personal Universal Controller:

The goal of this project is to provide a ubiquitous control of appliances in the environment from a mobile device. Here it is required to generate the graphical user interface on the fly in the mobile device. So the device needs to incorporate functionality to download the XML based functional specification from the appliance to be controlled and develop the user interface from that [4]. This downloaded specification is the functional description of the appliance rather than layout information. This is an interesting approach and is different from the project that this paper discusses. However it requires the mobile device to incorporate the capability of downloading and generating the GUI. This becomes a bit challenging, as memory is the key issue in hand held devices.

There are several other XML-based approaches that have been tried over the years for reducing the time and cost associated with the GUI development. However most of these approaches are domain specific and could not be considered as universal. They target either specific devices or specific platforms.

3. Need for a rule based approach

This paper proposes a Rule Based approach. It identifies a mechanism that allows defining the GUI once and then merely developing rules in a specific way that would help the underlying framework to automatically generate the code for a particular platform. The advantage of the rule based system is that, the rules are very close to the language they are written for and could let the framework generate code that uses most features of the language.

It reduces the development time exponentially. Though it requires some effort to develop new set of rules, however, once written, the framework is ready to transform any definition into that particular language using a language specific code generator. So it proves to be a one-time effort. Eventually the rule file and the code generator could just be a part of the underlying language that ships with the language and could be plugged in to the framework. The solution proposes an architecture that makes it possible to have a single universal definition of the graphical user interface and generate output in any language for any platform.

The following section discusses the architecture of the proposed framework and attempts to identify all the moving components of it.

4. Architecture

The GUI must be defined using MDML (multi-device markup language) schema that the framework uses. The MDML is a swing-based schema. The reason for this is that Java Swing incorporates a very rich set of widgets and layouts. Extremely sophisticated application can be developed using the Java Swing. MDML being Swing based would allow developing such a sophisticated GUI, thus allowing the framework to use all the features available for the desktop application development. However the framework would work around the problem of generating code for the platform that has limited set

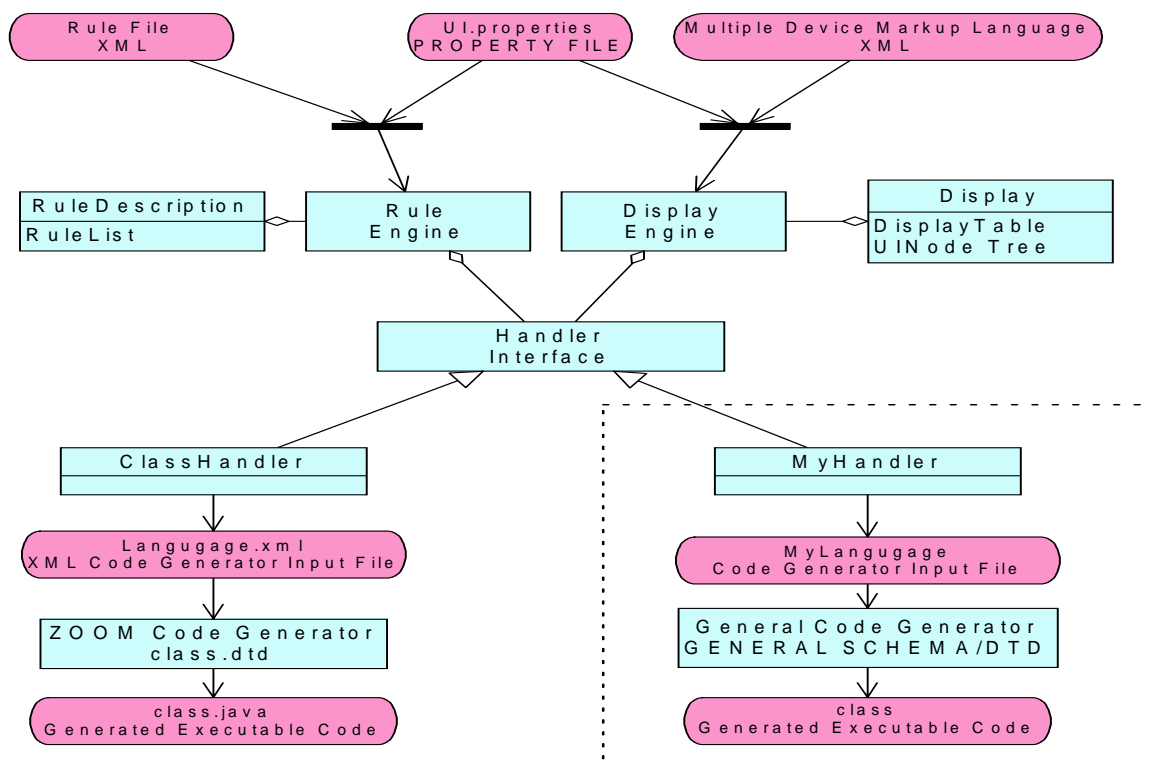
of features as compared to the Swing API. This framework uses this MDML file after validation and produces the necessary code.

The framework mainly consists of four parts:

1. Display Engine: Parses the input MDML file and processes it.
2. Rule Engine: Parses and validates the language specific rule file.
3. Handler: Processes the output of rule engine and the display engine, and generates an XML file that adheres to the schema dictated by the code generator.
4. Code generator: Produces the source code for the language.

Below is the complete description of the architecture for the framework

Framework Architecture



4.1 Display Engine

The input to the display engine is an MDML file that must strictly adhere to the schema mentioned by the UISchema.xsd file. Theoretically, MDML file can define the most sophisticated the graphical user interface that could be produced by the system. As mentioned earlier, MDML is swing based and hence allows the user to define a GUI using all the features available for the desktop applications. It is the purpose of framework then to suggest alternative ways to support the functionality if the output language is not able to support these features, thus migrating the responsibility a level down to the machine from the user.

The display engine is responsible for reading the initial MDML file into memory and changing layout as described by the rules. The display engine uses Document Object Model (DOM) to initially capture the MDML file into memory. Every tag is translated into a User Interface Node (UINode). All display engine objects - component(s), bargroup(s), container(s), and layout(s), will inherit from the UINode and thus creating a UINode Tree. This tree is the object representation of the MDML file for the internal use of the system.

The object representation of the MDML file is a language-independent GUI representation. The tree representation of the MDML file is useful for various processing reasons; however the look up for an object will involve tree traversal mechanism and will make the process time consuming. To support faster look up, each of these UI nodes are stored in a hash table by their respective unique keys.

4.2 Rule Engine

The input file to the rule engine is a rule file - rule.xml file. This rule file is divided into three distinct sections:

1. **Profile:** This section typically educates the framework for the files to be imported, profile details such as the output language and any back end resources to be used.
2. **Tag:** The tag section aids in allowing the framework to generate the code that is language specific. This section describes the corresponding set of widgets to be used for a specific set of widgets in the MDML file. It also defines the composition rules in order to house the widgets in a top-level container. Besides specifying the counter set of widgets to be used it also lets the designer specify the initialization method, constructor details and the method name by which the widget should be added to the container.
3. **Event:** The event section describes various kinds of events that could occur on the application. This section allows mapping various data manipulation techniques like the mouse event on the desktop application mapped with a pen event on the PDA.

The rule engine performs the validation of the rule file against the specified schema and is responsible for parsing the rule file. The rule files for Swing and SWT are already developed and exist on the project website [8].

4.3 Handler

As mentioned earlier, the object representation produced by the display engine is language independent. The language specifics are mentioned in the rule file mentioned above. It is the responsibility of the handler module to operate on the language independent object representation of the MDML definition and produce a language specific XML representation of the GUI with the help of the rules mentioned in the rule file. This is called as the language file and it adheres to the schema dictated by the code generator. It should be noted however that the language file is for the internal use of the application and will not be persisted.


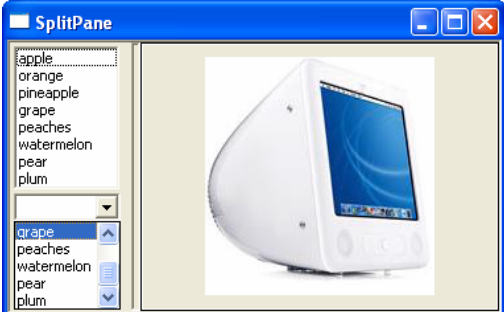
4.4 Code Generator

The code generator takes in the language file produced by the handler and generates the code. The code generator used by this project was developed for the ZOOM [7] project. The code generator is loosely coupled with the framework and hence a different code generator could be used if required. This will involve tweaking the handler module to produce the language file binding to the specifics of the new code generator. Since all the manipulation of the MDML file is done on its object representation, the handler is not bound to produce only XML file as required by the zoom project. It could produce a completely different representation for the purpose. Loose coupling of the code generator is a very big plus for the framework as it could be extended very easily.

5 Example

Below are the examples of two applications. The rules for Java Swing and SWT have already been written and only the MDML files are required to be developed which are shown in the application..

5.1 Application 1 – Split Pane

<pre><?xml version="1.0" encoding="UTF-8"?> <Start name="SplitPane" > <Window title="SplitPane" name="SplitPane" visible="true" show="true"> <SplitPane right="right" left="left"> <Panel name="left"> <BoxLayout axis="1"/> <List> <ListItem element="apple"/> <ListItem element="orange"/> <ListItem element="pineapple"/> <ListItem element="grape"/> <ListItem element="peaches"/> <ListItem element="watermelon"/> <ListItem element="pear"/> <ListItem element="plum"/> </List> <ComboBox event="Action"> <Item element="apple"/> <Item element="orange"/> <Item element="pineapple"/> <Item element="grape"/> <Item element="peaches"/> <Item element="watermelon"/> <Item element="pear"/> <Item element="plum"/> </ComboBox> </Panel> <Panel name="right"> <FlowLayout/> <Image imagesrc="sample.jpg"/> </Panel> </SplitPane> </Window> </Start></pre>	 <p style="text-align: center;">Swing Split Pane</p>  <p style="text-align: center;">SWT Split Pane</p>
---	--

MDML Definition

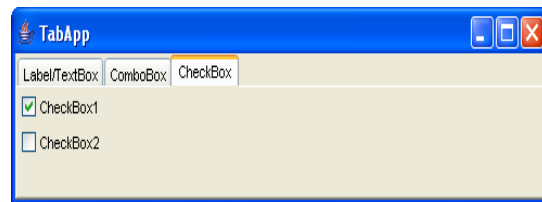
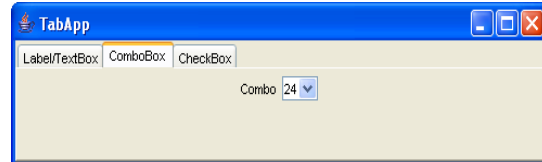
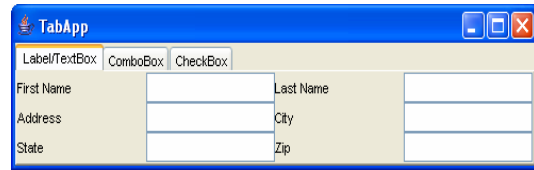
5.2 Application 2: Tabbed Pane

```

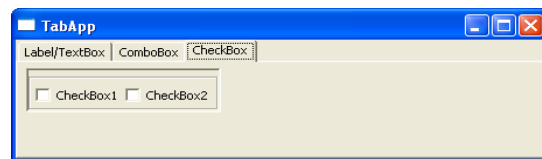
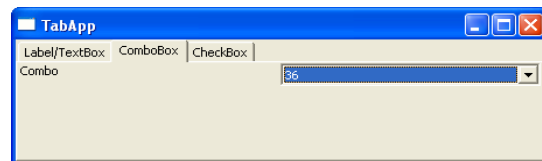
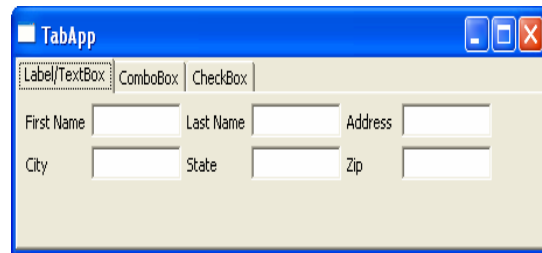
<?xml version="1.0" encoding="UTF-8"?>
<Start name="TabbedApplication" >
  <Window name="TabApp" title="Tabbed
Application" visible="true" show="true">
    <TabbedPane>
      <Tabs text="Label/TextBox">
        <GridLayout rows="3"
          columns="2" icolumns="6"/>
        <Label text="First Name"/>
        <TextBox type="text"/>
        <Label text="Last Name"/>
        <TextBox type="text"/>
        <Label text="Address "/>
        <TextBox type="text"/>
        <Label text="City"/>
        <TextBox type="text"/>
        <Label text="State"/>
        <TextBox type="text"/>
        <Label text="Zip"/>
        <TextBox type="text"/>
      </Tabs>
      <Tabs text="ComboBox">
        <FlowLayout/>
        <Label text="Combo"/>
        <ComboBox>
          <Item element="24"/>
          <Item element="36"/>
        </ComboBox>
      </Tabs>
      <Tabs text="CheckBox">
        <GridLayout rows="3"
          columns="1" icolumns="2"/>
        <Group name="GroupCheck">
          <CheckBox text="CheckBox1"/>
          <CheckBox text="CheckBox2"/>
        </Group>
      </Tabs>
    </TabbedPane>
  </Window>

```

MDML Definition



Swing Tabbed Pane



SWT Tabbed Pane

6. Event Model

The event model is general and shares the Swing naming convention. In this framework there exist six basic event types – action, focus, selection, mouse, mouse-motion, and window. The user of the framework is responsible for implementing the functionality or classes for these events. The framework requires the name of these

classes to be specified in the rule section. Each event type tag contains a <Field> tag. The <Field> tag contains two attributes - name and method. Name specifies the variable name and method specifies the method to use with the components. Under the <Field> tag the user must specify the type. The <type> tag specifies the class that the backend functionality must implement. Also, if the functionality is in a different package or subsystem, that package or subsystem must be specified in the <import> section of the profile.

7. Future Work

Mobile Profile

As stated earlier the goal of the project is to produce a GUI for multiple platforms and devices. This includes not only the high-end desktop devices (Desktop Profile) but also the low-end mobile devices. This section discusses extending the framework to support low-end devices using the MIDP Profile.

To support the mobile profile (J2ME) the display engine must be enhanced to produce an alternate layout from the original MDML file. There are two options to do this, let the user produce a separate MDML file for the mobile profile, or let the system do that by adding new rules. In other frameworks like UIML [2] the first option is used to generate GUI's across profiles. In this framework, the latter is desired, since the rule specification contains the toolkit characteristics.

There are two approaches to dynamically convert the MDML file into the one appropriate for the low-end devices. The first approach uses direct widget mapping technique. All widgets and layouts that can be defined in the MDML are mapped to the widgets of the MIDP API or any other API that is to be used. The problem with this approach is that in order to map all the widgets defined in the MDML schema to the low end API, several MDML widgets and layouts will map to a single or no widgets at all in the MIDP profile.

The number of widgets in the MDML clearly out numbers the widgets in the MIDP API and there is no one to one mapping; instead there is a group of widgets in the MDML that map to the same widgets in the MIDP set. This indicates that one or more functionalities in the MDML will be delivered using the same MIDP Tag. This approach is very naïve.

Another drawback is the limited screen size of the low-end device. It is absolutely impossible to display a sophisticated desktop GUI on a single page/screen of the mobile devices. So it is obvious that the single page desktop application must be divided into multiple pages on the mobile device. The MDML schema can provide a way to hint the rule engine about the page separations but there is no exact way to dynamically create these page separations. This could lead to a complete disaster. For instance, consider a simple desktop calculator application. It is impossible to map the complete application on the display screen of the mobile device. This mobile device could show six to eight widgets on a single screen as opposed to the 41 widgets that shows up on the desktop application. Thus it is clear that this application needs to be separated into multiple pages. If the rule engine is allowed to perform a naïve break down, then it will show a few

buttons on one page and a few buttons on the other pages. The naïve mapping does not break the application logically, thus making it difficult to use.

Though the application only demonstrates the problems with widgets, the same applies to layouts and other unsupported functionalities like the menu bar.

The above examples demonstrate major difficulties that must be addressed in order to make a sensible and usable transition from a desktop application to a mobile application. It clearly indicates that the rule engine below could be naïve and rendering a complete useless application so the framework must incorporate reasonable amount of intelligence.

The above problems could be summarized as below:

- Number of widgets in the MDML out numbers the widgets in the MIDP.
- Logical elements are not grouped together

The solution lays in *shrinking* [2] the desktop application. Instead of mapping the widgets naively, an attempt must be made to map the functionality as compared to the display in a way such that it reduces the number of widgets that are required for display. In this approach, the intelligence is incorporated inside the rule engine to produce the desired output. There are a number of intricate details about the device that the rule engine profile must take into consideration:

- Size of the MIDP display.
- The widget list that it supports.
- Constraints under which the rule engine should work.
- Other details such as the support for color and other GUI applications

The process that is described below should be used iteratively until the application could be easily displayed on the mobile device or the application could not go any further reduction. The following are the list of steps that the application should iterate through:

- a. Chunking
- b. Reduction
- c. Smart Mapping
- d. Elimination

a. Chunking [7]

In the first step the application needs to be separated into multiple logical pages. This could be determined from the MDML support tag known as the break point which is used to hint to system for a logical break. So the chunking process should mark the application into these logically separate parts and see if further reduction is necessary.

b. Reduction [7]

The text articles are way too long to be transported to the mobile application. It is surely possible to reduce it. There are various automatic text reduction algorithms that could work around the problem based on various things and reduce the text to some key words that would merely give the gist of the article. A good way is to take the help of the publisher of the article and reduce it. It is certainly not possible to shrink it to only key words, so some of them could be a link to a detailed description optionally that could be directed to but only if the reader wishes to.

c. Smart Mapping

This is the most interesting and important thing that should be understood very well. As mentioned earlier it might not be possible to display all the forty-one widgets in the calculator application on the same page. It is now required to reduce the number of widgets. Given a thought the number widgets could be mapped to a single widget – the text box. Also the four operation widgets could be mapped with just one widget – the command button thus transforming the application into a three widget application from forty-one that could be easily displayed on a single page. It might not always possible to reduce an application so drastically, but it is always possible to map a large number of widgets to a fewer one which helps in reducing the number of widgets to be displayed.

d. Elimination

If the image is not giving any information like a logo than it could be eliminated, as it would just prove to be an overhead in the mobile application..

This process could be iteratively used to shrink the application. Also there are specialized mobile web browsers to aid in displaying the images if required with a technique known as flip zooming [7].

7 Conclusion

Many projects take the approach to build an XML User Interface by tightly coupling XML tags to components or methods (i.e. THINLETS, LUXOR). This project is loosely coupled since the rule specification provides the mapping between XML tags and components. With relation to a desktop profile, <MENUBAR> means a menu bar will be attached to the given window or dialog, yet in a mobile environment menu bar do not normally exist. This framework will allow the end user to change the component mapping through XML rules without modifying the implementation.

Another benefit of this framework is that a developer is not tied to one code generator. Although, this project uses the ZOOM code generator, one could develop an entirely new Handler from the *Handler* interface. The new Handler could produce the desired input for another generalized code generator or use a XSL Transformation to transform the *Language file* into a valid input for a new code generator. Lastly, with this framework, the ZOOM model of separation between User Interface, behavior, and structural is proven for the User Interface. The UI model can be generated independent of any behavior or structure.

Using this framework requires thinking about design first and code later since the code is automatically generated. Unlike the traditional approach code and run the design is extremely import in this framework since the idea is it can be used on multiple platforms and devices with varying degrees of richness and limitations. Also, to the novice, it might appear that this framework requires a lot of overhead, rule specification and XML description to complete an application. But, once the rule specification is written generally it doesn't have to be changed unless the application rendered doesn't conform to the initial design.

References:

- [1] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, eds, Extensible Markup Language (XML) 1.0, W3C Recommendation, 10 February 1998, www.w3.org/TR/1998/REC-xml-19980210.
- [2] Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, Jonathan E. Shuster “UIML: An Appliance-Independent XML User Interface Language”, <http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html>
- [3] Mozilla, XUL Language Spec, *second draft*, 25 February 1999, www.mozilla.org/xpfe/languageSpec.html.
- [4] Automatically Generating Interfaces for Multi-Device Environments, Jeffrey Nichols and Brad A. Myers, <http://www-2.cs.cmu.edu/~jeffreyn/controller/nichols-multi-device-ws-proposal.html>
- [5] The Joy of XUL, Peter Bojanic, <http://www.mozilla.org/projects/xul/joy-of-xul.html>
- [6] Multimodal Interactive User Interfaces for Mobile Multi-Device Environments, Robbie Schaefer, Wolfgang Mueller, Paderborn, Germany, <http://www-users.cs.umn.edu/~terveen/ubicomp2003/schaefer.pdf>
- [7] WEST: A Web Browser for Small Terminals, Staffan Björk, Lars Erik Holmquist and Johan Redström, Ivan Bretan, Telia Mobile AB, Rolf Danielsson, Telia Research AB, Jussi Karlgren and Kristofer Franzén, <http://www.sics.se/~franzen/Artiklar/UIST99/west.pdf>
- [8] MULTIPLE DEVICE MARKUP LANGUAGE-A RULE APPROACH, PAUL D JOHNSON AND JINESH PAREKH,**
[HTTP://SHRIKE.DEPAUL.EDU/~PJOHNSO2/FINAL.DOC](http://shrike.depaul.edu/~pjohnso2/FINAL.DOC)