Generative Approaches for Application Tailoring of Mobile Devices

Victoria Davis and Jeff Gray
Dept. of Computer and Info. Sciences
University of Alabama at Birmingham
Birmingham, AL USA 35294-1170
Phone: (205)-934-2213
{davisvm,gray}@cis.uab.edu

Joel Jones
Department of Computer Science
University of Alabama
Tuscaloosa, AL USA 35487-0290
Phone: (205)-348-6363
jones@cs.ua.edu

ABSTRACT

The popularity of mobile devices has propelled the development of many useful location-aware applications. However, the heterogeneity of mobile devices necessitates that the software be customized and tailored for each device. The research described in this paper demonstrates the possibilities of generative programming applied to application tailoring. This is done in order to assist in porting software to specific devices without manually rewriting code. The Java 2 Micro Edition (J2ME) is an integral part of the application tailoring solution. Many mobile devices are capable of using J2ME, but require the code to be packaged specifically to run in each different mobile environment. J2ME applications alone are not sufficient for porting the code to different mobile devices.

The first solution that will be presented uses a specifically structured VoiceXML file as input to an XSL transformation. The transformation produces J2ME source code. Java servlets are used to compile the resulting code and package it with respect to a specific device. This first solution works well for users who have programming experience and are comfortable with editing XML files. However, a different solution is needed to enable users with limited programming experience to specify the essential properties of the mobile application.

A second solution to application tailorability uses a metamodeling tool (we use the Generic Modeling Environment – GME) to create a domain-specific modeling language. This environment allows an end-user to capture the essence of a design in a notation that is familiar to the users. From the specified models, an application can be generated directly from a model interpreter. The modeling approach provides a higher-level of abstraction, which removes the user from the accidental complexities regarding the details of the mobile application implementation. A case study is presented that enables a restaurateur to create an online menu for use on several different mobile devices.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
43rd ACM Southeast Conference, March 18-20, 2005, Kennesaw, GA, USA. Copyright 2004 ACM 1-58113-000-0/00/0004…$5.00.

1. INTRODUCTION

This paper describes two approaches for providing application tailoring. Application tailoring solutions are needed due to the increase in availability and the popularity of mobile devices. According to CTIA there are 171.2 million wireless subscribers in the United States [4]. The increase in mobile devices drives a desire for more applications to run on the devices. According to Adrian Friday, "For a PDA to become accepted as a viable alternative to existing personal organization tools, it must be equally convenient and reliable."[1] These devices become more convenient when software applications with rich functionality are developed for them. In addition to a PDA, the same can be said for other mobile devices, such as cellular phones, notebook computers and Blackberry's. Two important differences among wireless devices are the screen size and the operating environment. These two differences require the application to be tailored specifically for each device.

The differences that exist between the screen size and the operating environment create the need for different artifacts that are used to package the application for each device. Artifacts are files that are necessary to deploy the application on different devices. A notebook computer, which has a large screen size, can use HTML files that have a rich visualization of the provided content. Due to the smaller screen display of other mobile devices HTML applications are not suitable. A different version of the application will have to be written to target other mobile devices, each device having their own packaging requirements. The same application for a PDA might require the application to be packaged in .PRC files that are uploaded to the device. Similarly, a cell phone would require a .JAD/.JAR file to be sent to the device, and a Blackberry might need the same application packaged in a .COD/.JAR file.

Model-driven application tailoring provides a solution that addresses these issues by creating domain-specific models and abstracting away the complexities associated with each device. End-users with limited programming skills will have the ability to create their own applications for the devices. The additional applications that can be created by the end-users can increase the convenience of the mobile devices. This case study will use the domain of a restaurateur who wants to make an online menu available to his potential customers who use various mobile devices (see Figure 1, which shows a menu for Ice Cream that is available across multiple devices).

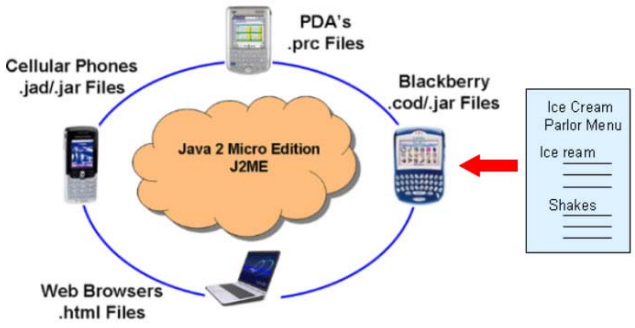

Figure 1. Online menu tailored for many clients

One commonality among the many devices is the capability of running the Java 2 Platform Micro Edition (J2ME), which is targeted to consumer electronics and embedded devices [5]. J2ME solves part of the application tailoring solution by addressing the needs of the operating system and screen sizes. The Java HQ runtime environment is available for mobile devices, such as PDAs. The screen size issue is solved by using J2ME's high-level APIs, which allow a device to choose how to display objects such as buttons and text onto the screen. There is also a low-level API that is available, but it requires a programmer to be responsible for everything that is displayed on the screen. High-level APIs remove the screen size programming dilemma from the programmer and places the responsibility on the mobile device.

However, J2ME is not the entire solution. Many devices are capable of using J2ME, but a device that is labeled as J2ME compatible does not necessarily mean that it is able to run J2ME programs packaged as .class files. Most devices require J2ME programs to be packaged for their own environment outside of the mobile client. The fact that J2ME is not sufficient alone can cause frustration for potential developers targeting mobile devices.

The next sections will describe the two application tailoring case studies that are based on techniques from generative programming [12]. Section 2 describes a generative approach that synthesizes applications from specifications contained in an XML file. Section 3 presents an alternative that is based on model-driven principles using the GME metamodeling tool. Summary and concluding remarks are contained in Section 4.

2. TAILORABILITY THROUGH XML

The first approach utilizes VoiceXML (VXML) [6] as input to an XSL translation. This section summarizes these two technologies and provides an example of how they were used in the application tailorability problem. This case study demonstrates how a restaurateur with prior knowledge of XML can create a VXML document for an online menu application. From the VXML specification, a complete mobile application is then generated. The mobile client will be presented with a list of application options. The options are an HTML page containing the menu, or a downloadable Palm application. The end-user (the restaurateur) is responsible for creating the XML configuration file based on a particular structure for the restaurant menu. Java servlets handle application tailoring. Many of the accidental complexities associated with application tailoring are abstracted away from the restaurateur. In the XML-based approach, application tailoring occurs at runtime.

2.1 VoiceXML

VoiceXML is the notation used for the first solution of the case study. VXML is intended to enable users to interact with the web through voice commands. It is also used for applications providing automatic answering services. VXML provides a consistent structure that has been standardized by the World Wide Web Consortium [6]. The case study is not a voice application, but the structure of VXML is used as the foundation for the input file that specifies the details of the mobile application.

In the VXML file, the restaurateur defines the structure of the restaurant menu through XML tags. The tags represent a query and response for specific menu items. In the generation to Java, the VXML <prompt> tags become questions to be presented to the mobile client. The response of the client is captured in a list of VXML <item> tags. Menu items can be added or deleted as necessary by adding or removing <field> tags. One VoiceXML document provides a single input for many translations. Listing 1 shows the structure of a menu item in VXML.

```
<field name = "container">
 <prompt>Do you want a cup or cone.</prompt>
 <prompt>Say cup or cone.</prompt>
  <grammar type="application/srgs+xml" root="r2" version="1.0">
   <rule id="r2" scope="public">
    <one-of>
     <item>cone</item>
     <item>cup</item>
    </one-of>
   </rule>
  </grammar>
 <field>You said <value expr="container" /></field>
</field>
```

Listing 1. VoiceXML to request ice cream container

2.2 XSLT Transformation of VoiceXML

To translate VXML to J2ME, an Extensible Stylesheet Language Transformation (XSLT) can be applied to the VXML document. XSLT is a transformation language that consists of a set of rules for transforming a source tree into a result tree [7]. The XML Path Language (XPath) [8] provides access to the XML tree structure to navigate and access the data in the XML file. XSLT allows output to be directed to a plain text file, which in this study is J2ME source code. Listing 2 shows a portion of XSLT that transforms the <field> tag of the VXML document into J2ME code. Each time a <field> tag is found in the tree, the corresponding XSLT code will generate the J2ME code. The generated code (see the last four lines of Listing 2) is parameterized by the information matched in the XML <field>. Several XSLT translations have to be written to produce the other artifacts necessary for packaging and tailoring the application. The artifacts needed for a cellular phone application are the Manifest file (MF), and the Java Application Descriptor (JAD) file. The Java 2 Platform Micro Edition Wireless Toolkit creates these files automatically if the toolkit is used. It is necessary to compile and package the application from the command line for this solution.

```
<!--Each menu item is taken from the first prompt in each <field>-->
<xsl:for-each select="field">
 <xsl:text> menu.append("</xsl:text>
   <xsl:value-of select="prompt[1]" />
 <xsl:text>", null);&#10;</xsl:text>
</xsl:for-each>
<xsl:text> menu.addCommand(orderCommand);&#10;</xsl:text>
<xsl:text> menu.addCommand(exitCommand);&#10;</xsl:text>
<xsl:text> menu.setCommandListener(this);&#10;</xsl:text>
<xsl:text> mainMenu();&#10;} >}&#10;</xsl:text>
```

Listing 2. XSLT translation of <field> nodes to J2ME

2.3 Servlets

The Apache Tomcat server [9] is used in our implementation of this case study. Specifically, the Xalan-Java XSLT translation engine and the Xerces-Java XML parser were the key parts of Tomcat that were used [10]. Java servlets deployed on Tomcat provided the necessary infrastructure for the application tailoring experiment.

```
menu.append("Do you want a cup or cone.", null);
menu.append("Select a flavor of Ice cream.", null);
menu.addCommand(exitCommand);
menu.setCommandListener(this);
mainMenu();
```

Listing 3. Generated J2ME code

An HTML page must be created to present the mobile client with the generated application options. When an option is selected from the HTML page, it sends a request to the first of three servlets necessary for application tailoring. The first servlet takes in the request for a particular application. It passes control to a second servlet which is responsible for that particular translation request. The second servlet performs the majority of the tailoring for this solution. The servlet accesses the command line of the operating system and creates the directories that are needed for the tailoring process. This second servlet invokes the translation engine (Xalan) and applies the XSLT translations. Listing 3 shows a partial result of the translated J2ME code.

Three translations are necessary for translating VXML to all the files necessary for application tailoring. The translations needed are: one for J2ME, one for the MF file and the last for the JAD file. After all of the translations have completed, the servlet accesses the command line again to finish the tailoring process. To produce an application for a mobile phone, the servlet must compile, pre-verify, and generate a Java Archive (JAR) file of the J2ME code. At this point there has been enough tailoring to deploy the application to a mobile phone. However, to produce a Palm application, additional steps must be taken. The servlet must start the Mobile Information Device Profile (MIDP) for the Palm OS [11] converter. This application can run from the command line to convert the JAD file to the Palm OS Resource Collection file (PRC). Once those steps have completed, the application is ready to be deployed on a Palm device. Control is then passed to a third servlet. The third servlet sends the application in a TCP stream to the client.

If the first servlet gets a request for an HTML file, control is passed to a different second servlet. This alternative servlet is similar to the servlet that handles a Palm request, but does not have to perform all of the same steps for packaging. The same VXML document serves as the input for the HTML translation, but a different XLST translation would be applied. This XSLT document will output HTML code. Control would then be passed to a third servlet to deliver a link to the generated HTML file to the mobile client.

The benefit of this approach is that the essence of the application is captured in the VXML file, which can be used to generate the lower-level implementation and configuration. To evolve and change the application requires modifications to the VXML file, rather than code. A single change to the VXML file has the capability to affect multiple artifacts for various devices.

3. MODEL-DRIVEN TAILORABILITY

The second approach applied to the case study uses a metamodeling tool called the Generic Modeling Environment (GME) [2]. The GME is used to create a domain-specific modeling language and environment. The end-user will be responsible for building a model of the menu in a customized modeling environment provided by the GME. From a model, a model interpreter can be used to generate all of the artifacts needed for application tailoring. After the interpreter has been invoked the generated tailored applications are ready to be made available to the mobile client. The metamodeling approach provides a higher-level of abstraction than the VoiceXML approach. In the GME model, application tailoring occurs at design time.

3.1 A Metamodel for Application Tailoring

A metamodel specifies a visual modeling language for a specific domain. In this section, a metamodel is described that represents all the concepts needed to build the restaurateur online menu. Using the GME, three elements are needed to produce the metamodel for this domain. They are a model, an atom and a connection. The model element is used to represent a welcome message and also a menu item. The atoms represent the choices contained within the menu item model. The connection entity specifies the relationship between the models and the atoms.

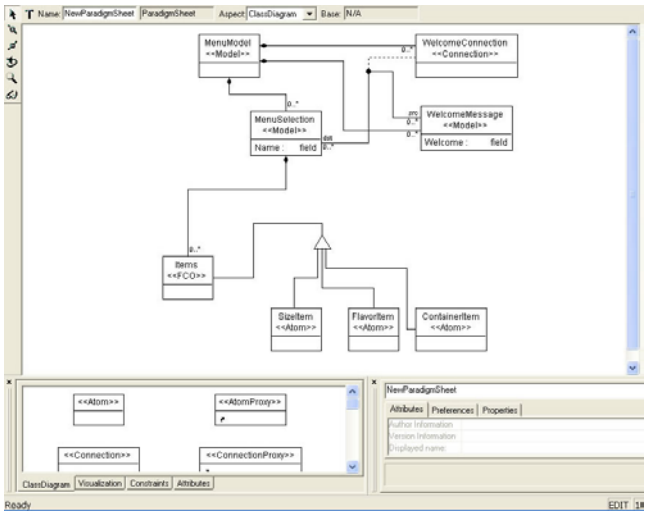

Figure 2. The restaurant metamodel in GME

Once these elements are created and placed in the metamodel, additional information can be added to them by attaching attribute information. The attributes are needed because they contain information that is vital to describing particular details about the element. The welcome message has an attribute that allows for a custom welcome message to be displayed. The menu item has a field attribute to describe the name of the menu item. Custom icons are created that represent the elements of the metamodel. They are added to give the modeling environment a look that is representative of the particular domain. Once the metamodel has been constructed it must be interpreted to produce the modeling environment for the restaurateur domain. The GME provides existing capabilities to generate a modeling environment from the metamodel. Figure 2 shows the metamodel for this case study.

3.2 Applying the Restaurant Metamodel

The restaurateur is not responsible for creating the metamodel or the translations. The restaurateur constructs instances of the metamodel to configure the online menu. All the elements needed for the online menu are visible on the screen and can be added to the model by dragging and dropping them on to the screen. The first thing the restaurateur must do is add a welcome message to the model. Then, for each food item that needs to be on the online menu, a menu item element must be added to the model. When a menu item has been placed into the model it can be opened by double clicking on the menu element, which will open another window where objects that describe the menu item are located. These objects can be placed into the menu element and named. Figure 3 shows the items contained within the "IceCream" menu item. Figure 4 shows how the model of an online menu would be displayed in the GME. Such models can be constructed by those who have little or no knowledge of programming languages and the details of mobile application development.

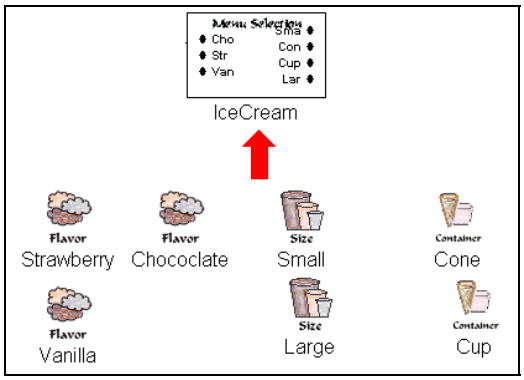

Figure 3. Contents of a menu item

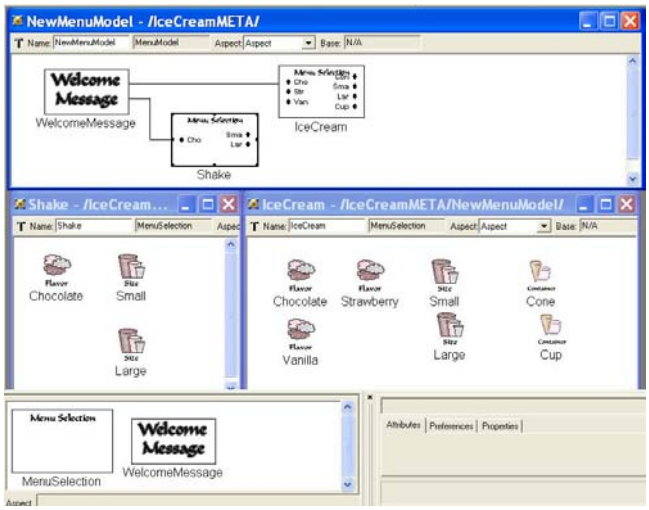

Figure 4. The domain-specific model in GME

3.3 A Model-Driven Tailorability Interpreter

The GME provides several ways to access the data from the model. One of these is the Builder Object Network (BON), which provides access to the internal representation of the model through C++ objects. An interpreter in the GME can be written in C++. The model-driven application tailoring solution using the GME is similar to the approach adopted in Section 2, except that application tailoring in GME is contained in the interpreter, and not handled by servlets.

The interpreter traverses the model and generates all of the required artifacts necessary for the application to run on the specific device. In this study, the specific device is a Palm handheld. The interpreter generates the J2ME code as well as the MF file and the JAD file. Listing 4 shows the C++ code that generates a portion of the J2ME code. The method iterates over all menu selection elements in the model (e.g., "Ice Cream" and "Shake," as shown in Figure 4), and generates the J2ME code to add an item to the application menu. In addition to traversing the model and generating the required artifacts (e.g., Java code), the model interpreter also interacts with the file system and builds the necessary directories needed to package the application.

The end-user runs the interpreter once the model has been constructed to represent the essence of the application that is to be sent to the mobile clients. The interpreter must contain transformation code for every artifact needed in the tailoring process. The interpreter then runs batch files that it generates. These batch files compile, pre-verify and create the JAR file for the application. Then it executes the MIDP for the Palm converter. This is accomplished by accessing the command line from inside the C++ code of the interpreter. Once this is finished the application is ready to be deployed to the Palm handheld. Tailoring is the result of running the interpreter to produce the applications necessary for the different mobile devices. An interpreter must be written for each target device. In contrast to the run-time tailoring provided by the XML-XSLT approach, application tailoring in the GME is done at design and compile time.

```
//=====Processes the MenuSelections for the Main Screen of Menu Choices
void CComponent::ProcessMenuSelectionScreen(CBuilderModel * r) {
 ASSERT(r->GetKindName() == "MenuSelections");
 CString message;
 r->GetAttribute("Name", message);
 outf << " menuitem.append(\"" << message << "\", null);" << endl;
}
```

Listing 4. C++ code to generate J2ME

It should be emphasized that the metamodel and the associated interpreter are not built by the end-user. These are constructed by modelling experts who use GME to build a domain-specific modelling environment. The end-user works in this environment to create models of their specific application, such as those found in Figure 3 and Figure 4.

4. CONCLUSIONS

The results of the case study show that it is possible to tailor applications for specific devices without involving low-level changes to source code. Through generative programming, a single higher-level specification can be synthesized into multiple lower-level implementation artifacts. For example, configuration information captured in an XML file or a graphical model can be used to generate multiple files to configure a Palm application (e.g., the required Java, JAD, and MF files). The files are generated automatically by the translator, rather than coded explicitly. Application tailoring is abstracted away from the end-user and is removed from the mobile client. Many of the accidental complexities that arise when building different mobile applications are taken care of when generative approaches are used. Simple things that get overlooked, such as directories that need to be created and file names that don't interfere with each other, are a few of the details that are abstracted away.

The case study demonstrated two approaches showing the power of abstraction in removing accidental complexities of the implementation. In the example of the VoiceXML and XSLT approach, the VoiceXML file contained 32 lines of specification, but generated over 200 lines of Java code. The second approach, based on the GME model, generated the same Palm application, plus additional batch files for command line processing (over 250 lines of Java code). Generative programming approaches eliminate the need to rewrite the code manually, which can improve productivity by allowing the essence of a problem to be the primary focus. Additionally, programmer errors caused by manual rewriting of low-level code can be reduced through generative techniques by putting the burden on the automated translators.

Both of the approaches presented in the paper give the end-user (who may have limited programming skills) the ability to create their own mobile applications. The XML approach provides a run-time application tailoring solution by running the tailoring on a server. This allows a tailoring solution to be created when it is requested by a mobile client. It requires the end-user to be familiar with XML, but offers fresh content every time a change is made to the VXML file. The details of tailoring are spread out over a variety of mediums, such as the servlets, XML, XSLT.

From our evaluation, the domain-specific modeling approach provided by GME offers a higher-level of abstraction than the XML/XSLT approach. With domain-specific modeling, the end-user does not have to understand VXML. In the modeling domain presented in Section 3, the restaurateur can build a restaurant menu with graphical elements that represent concepts that are more familiar to the domain (i.e., graphical icons that represent the items of a menu, rather than the XML tags that need to be modified in VXML).

5. ACKNOWLEDGMENTS

Portions of this research were funded from an NSF REU grant (CNS-0244156) entitled, "Pervasive and Mobile Computing."

Victoria Davis is now with Computer Technology Solutions, Inc., of Birmingham, Alabama.

6. REFERENCES

[1] Adrian John Friday, "Infrastructure Support for Adaptive Mobile Applications," PhD dissertation, Computing Department, Lancaster University, England, September, 1996, page 32.

[2] Akos Ledeczi, Arpad Bakay, Miklos Maroti, Peter Volgysei, Greg Nordstrom, Jonathan Sprinkle, and Gabor Karsai, "Composing Domain-Specific Design Environments," *IEEE Computer*, November 2001, pp. 44-51.

[3] Greg Nordstrom, Janos Sztipanovits, Gabor Karsai, and Akos Ledeczi, "Metamodeling - Rapid design and evolution of domain-specific modeling environments," *IEEE Engineering of Computer Based Systems (ECBS)*, Nashville, TN, April 1999, pp. 68-74.

[4] *CTIA – The Wireless Association*, CTIA 10 Nov 2004 http://www.ctia.org.

[5] *Java 2 Platform, Micro Edition (J2ME) overview*, Sun Microsystems 10 Nov 2004, http://java.sun.com/j2me/.

[6] *W3C "Voice Browser" Activity*, W3C, 10 Nov 2004, http://www.w3c.org/Voice/.

[7] *The Extensible Stylesheet Language Family (XSL)*, W3C, 10 Nov 2004, http://www.w3c.org/Style/XSL/.

[8] *XML Path Language (XPATH)*, W3C, 10 Nov 2004, http://www.w3c.org/TR/xpath/.

[9] *The Jakarta Site - The Apache Jakarta Tomcat*, The Apahce Jakarta Project , 10 Nov 2004, http://jakarta.apache.org/

[10] *Xalan-Java Overview*, The Apache XML Project, 10 Nov 2004 http://xml.apache.org/xalan-j/overview.html

[11] *MIDP for Palm OS*, Sun Microsystems, 11 Nov 2004, http://java.sun.com/products/midp4palm/

[12] Krzysztof Czarnecki and Ulrich Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000.