Reversing GUIs to XIML Descriptions for the Adaptation to Heterogeneous Devices

Giuseppe Di Santo  
Research Centre on Software Technology – RCOST  
University of Sannio  
Via Traiano, Benevento, 82100 Italy  

giuseppe.disanto@unisannio.it

Eugenio Zimeo  
Research Centre on Software Technology – RCOST  
University of Sannio  
Via Traiano, Benevento, 82100 Italy  
eugenio.zimeo@unisannio.it

Abstract
The spread of Personal Wireless Devices (PWDs) has raised the need to migrate existing applications to these new environments. Desktop applications often exhibit complex user interfaces and are too large and resource demanding to be executed on devices with limited resources without changing the application code. Current research efforts are mainly focused on Web applications whose user interfaces are specifically designed for multi-platform environments through platform-independent models. On the contrary, little effort has been made to support the migration of applications with component-based GUIs towards PWD environments. This paper presents a tool for reverse engineering Java GUIs through their transformations to XIML-based abstract descriptions. The resulting descriptions are used by the TCPTE framework to be rendered into different GUIs, which are dynamically adapted to heterogeneous devices on the basis of their profile communicated at request time.

Categories and Subject Descriptors
D.2.11 [Software Architectures]: Patterns. D.2.2 [Design tools and Techniques]: Software libraries, User Interfaces. D.2.7 [Distribution, Maintenance, and Enhancement]: Restructuring, reverse engineering, and reengineering.

Keywords
Reverse engineering, middleware, software architectures, graphical user interfaces.

1. Introduction
The highly distributed, heterogeneous, and mobile nature of Personal Wireless Devices (PWDs) is raising new challenges in software engineering that permeate the entire life cycle of application development for these new environments. Developers are often called to manage devices heterogeneity and constraints that are not alleviated by the use of virtual machines or web technologies. Such constraints demand highly efficient software systems in terms of computation, communication, and memory footprint, which are often overcome by off-loading some parts of a software system to other devices. The infrastructures of such technologies may thus lack certain services for reasons of efficiency and developers must make tradeoffs to address the computing constraints.

In [1], we proposed an approach based on library substitution to show graphical user interfaces of interactive applications onto devices with limited resources, while the application logic runs on a server. The approach consisted in implementing the half-object pattern [1, 2] inside the AWT package in order to separate the presentation layer of an application in two halves (server-side and client-side). While the approach is able to transparently migrate monolithic interactive applications towards client/server environments, by off-loading some application components on different devices, it is not sufficient to overcome the heterogeneity of devices, one of the most critical issues in the development of applications for PWDs.

Since heterogeneity demands for GUIs adaptation to a variety of hw/sw platforms, in this paper we intend to give an answer to the following question: how an existing GUI-based application can be adapted to be used on different devices with a small re-engineering effort? We can observe that a GUI for a PWD can be considered as a subset of widgets of its original desktop version, organized according to a layout more apt for the presentation on devices with limited screen sizes. Starting from this observation, the adaptation of GUIs to heterogeneous devices consists of three main actions: (1) selection from the original desktop GUI of widgets to show on PWDs; (2) mapping of the abstract interactive components, selected from the previous point, to concrete representations tied to specific graphical systems; (3) allocation in space and time of each widget in order to coexist with screen size constraints.

The first action is not simple if the starting point is an existing application to port on different devices. The critical aspect is the necessity of modifying the application source code to re-engineer the GUI for the presentation on a collection of devices. An abstract representation of existing GUIs can simplify the adaptation to different devices and can represent the first step towards automatic adaptation at run-time on the basis of device profiles and user preferences. Moreover, GUI abstract representations allow for handling meta-data, semantic annotations and inference in order to support reasoning during GUI rendering on the target device.

This paper proposes a novel approach which integrates the advantages of formal representations of UIs with a middleware organized according to the remote presentation model to allow
both lightweight and resource demanding applications to be completely or partially executed on heterogeneous mobile devices. As consequence, our approach to support migration of Java AWT applications to client/server environments exploits reverse engineering, transformations for abstract re-engineering, and incremental rendering of the reengineered GUI onto the target device.

We support these stages through a tool for reverse engineering and a middleware platform for the execution in a client/server environment. The middleware platform adopted is TCPTE, whose internals were presented in previous works [3].

In addition to the state of the art, TCPTE provides a novel approach that integrates the advantages of formal representations of GUIs with a middleware organized according to the remote presentation model to allow both lightweight and resource demanding applications to be migrated to a client/server environment. This way the application logic remains on a server while the user interface is shown on the client according to the features of its graphical system.

This paper is mainly focused on the analysis of the tool and its application to a case study. To this end, the rest of the paper is organized as follows. Section 2 describes the researches and tools proposed in the literature. Section 3 gives a brief overview of TCPTE and its architecture. Section 4 illustrates the technique and the tool to perform reverse engineering of existing GUIs to an XML-based meta-language for user interfaces. Section 5 shows how the tool supports the adaptation of user interfaces and how TCPTE is able to render abstract user interfaces to concrete GUIs on specific target devices. Section 6 discusses the results and highlights new research directions. Finally, section 7 concludes the paper.

2. Related Work

Developing applications taking into account some hardware constraints, such as CPU and memory, of heterogeneous personal devices is an open problem in software engineering. Some researchers [5] propose to re-factor desktop, monolithic applications through the adoption of a specific little language employed to separate the components used for the presentation from those used to process data. This schema conducts to the individuation of a separation line between client and server that is located in a point very similar to the one individuated by the remote presentation variant of the thin-client model. This model is well suited both to wireless communication, due to the limited number of interactions between client and server, and to reduced screen size of PWDs, due to the capability of mapping logical number of interactions between client and server, and to reduced screen size of PWDs, due to the capability of mapping logical GUI components onto different native ones.

Differently from [5] that needs a domain expert to separate the presentation logic from the business logic, other approaches aim to make transparent the separation by designing component frameworks that typically extend existing frameworks for graphical user interfaces. The most know approaches are Ultra Light Client (ULC), Classic Blend for Java, Thin-Client Framework (TCF), Remote Abstract Window Toolkit (RAWT).

All these frameworks do not easily support the development of thin-client applications on limited devices, since they do not tackle all: (1) low bandwidth and high latency that characterize wireless networks used on PWDs; (2) reduced size and resolution of PWDs.

Limitations of networks and screen features represent key problems for PWDs since they differentiate significantly PWDs from desktop computers and make it difficult to port even lightweight applications, such as office and personal management applications. These applications could be entirely executed by a personal device, but the limitation of PWDs’ screen size often imposes a restriction on the complexity of user interface. Moreover, devices heterogeneity requires software developers to implement different GUIs for different kinds of devices, due to the difference of screen sizes, graphical systems and interaction mechanisms and tools. A more interesting approach is the definition of abstract GUIs to render on different devices at runtime. The most used approaches [6, 7, 8] use XML-based languages for user interface definition. These approaches have been mainly used in web applications while less effort has been done for other environments. Therefore, the problem of porting existing applications from desktop environments to PWDs ones remains an open issue.

3. TCPTE overview

TCPTE was designed according to the remote presentation architecture to transparently distribute the main components of interactive applications, namely user interface and application logic. Therefore, resource-demanding applications can be decomposed moving the application logic on a server and the user interface on the wireless device. This division is performed at deployment time and is totally transparent to developers and end-users.

![Figure 1. TCPTE Layers](image-url)

The TCPTE architecture is shown in Figure 1. The application layer on the server side represents the application running on the server, whose output is displayed on a remote device. The application uses the components defined by the server side AWT Layer (in the following called AWTServer), which are proxies to graphical components stored on client side AWT layer (in the following called AWTClient): a method call on an AWTServer component is forwarded to the corresponding AWTClient component on the device.

This approach has two main advantages: first, during the development of an application the developer does not have to take into account the limitations of the device on which the UI is going to be shown, since the application logic will be executed on the server side; second, Java applications written using Java AWT can be easily ported on wireless devices. The communication layer deals with ad hoc serialization/deserialization techniques of objects transferred over the network. In order to improve the performances of the communication layer, the objects transferred between client and server are serialized in a binary format while the communication follows a one-way scheme (without
acknowledgment) for void method invocations and a synchronous two-way scheme for non-void method invocations.

Finally, the session layer deals with connection setup and maintenance between client and server, since for several reasons, such as out of range in wireless communication, connectivity between client and server could be temporary lost.

4. Reversing AWT GUI to XIML description

The approach we defined to reverse engineering existing GUI-based applications in order to simplify their transformations for the adaptation to different devices consists of three phases. The approach considers as first phase the transformation of GUIs from Java byte code to abstract descriptions coded in XML-based files. These files are modified according to the features of the potential devices that can be used to show the user interface. Each transformation will be used at rendering time to build the interface for the desired device.

This approach is not completely new. A similar approach has been proposed in [6, 9], but it is limited to the reverse engineering of HTML code and does not tackle the problem of GUI rendering onto different PWDs. On the other hand, the approach proposed in [10] follows the same flow of steps starting from step 2, since it tackles forward engineering of interactive applications for multi-device users, but the main focus of the paper is on the task model defined at design time in order to guide the generation of multiple GUIs, one for each kind of platform. It is worth noting that none of these proposals supports the transparent transformation of a monolithic interactive and resource demanding application (not web based) into a client/server one.

4.1 AWT2XIML

Among the XML-based languages proposed in the literature to describe user interfaces, XIML (eXtensible Interface Markup Language) [4] was chosen due to its flexibility, expressivity and completeness. However, other languages with similar characteristics could be employed.

Considering XIML as target language for the abstract descriptions of existing GUIs, we developed a tool, which is able to transform Java byte-code of interactive and AWT-based applications into XIML files.

The current implementation of the tool, called AWT2XIML, is able to analyze a jar file, containing the application binary code, and to display its content in a window panel. The developer is able to analyze the application classes by selecting them and by starting the analyzer. The analysis produces an XIML file that describes the widget structure according to a hierarchical organization of tags that reflects the same organization of the components’ structure.

Figure 2 shows an example of tree transformation. Starting from a widget called ColorApps that is a class extending Java.awt.Frame, AWT2XIML produces the tree of the original components represented as an XIML file. Figure 3 shows the source code of the ColorApps. When the class is instantiated, it produces a Frame that contains a Panel with three buttons (Blue, Red, Yellow). The resulted component tree is analyzed by AWT2XIML that in turn produces an XIML description of the widget. In the XIML description, each AWT component is described by an XIML presentation element labeled with a unique ID.

In the example, the class ColorApps is described by using the presentation element with id example.ColorApps_desktop. This element contains the presentation elements that describe the graphical components instantiating in rows 11 to 14 of Figure 3-a.

Figure 3. Java AWT source code

The parent-child relation of the component tree defined by the ColorApps class is preserved in the XIML description by describing the containers as XIML presentation elements that recursively contain other elements. So, in our example, the example.ColorApps_desktop element contains the XIML presentation element that represents the panel. This last element in turn contains the XIML presentation elements representing the buttons. Moreover, some components’ properties are stored in the XIML description, such as the layout of the container and the location of the components in the container.

The XIML code is the output of the first phases and it represents the input for the successive step.

5. Transformation and adaptation

The adaptation and rendering phases follow the reverse engineering phase. During the transformation phase, for each XIML widget description produced in the previous phase, the developer prepares multiple XIML descriptions, each one suitable for the platform that can be used to show the user interface (see Figure 4). These XIML descriptions represent the transformation rules to use at run-time for showing and adapting the widgets on the client.

In particular, the XIML description obtained in the first phase is shown by using another tool integrated in AWT2XILM. This tool permits the widget description to be modified in order to obtain multiple definitions of the widget. The description can be defined by deleting, moving or adding new XIML elements without
modifying the application code. By using the tool, the developer can make different descriptions of the same widget so defining the transformation rules for different target devices.

The transformation rules are implemented by the renderer which is interposed between the SC-Tree and the CC-Tree and located on the server-side (for performance reasons).

When a device connects to the server to access to a deployed application, the renderer, starting from the XML description, creates the apt TRTs by using the device profile and starts the application. When a widget is instantiated, AWTServer verifies whether the TRT for that component has been defined. If no transformation rule has been previously defined, the widget is left unchanged and each invocation of the structural method is directly propagated to the client. Otherwise, all the invocations are intercepted by the renderer that using the SC-Tree creates the equivalent CC-Tree.

6. Case Study: Bloof

The framework proposed in the paper was tested by porting onto a PDA a management tool for a database. The application chosen was Bloof (http://bloof.sourceforge.net/), a Java library for analytical processing of version control data. Bloof’s goal is to help developers, project managers and software engineers in comprehending the evolution of a software system.

Figure 6 shows the Bloof user interface for desktop platform (Figure 6-a) and its reorganized version running on: (1) an iPAQ Poket-PC (Figure 6-b) using the CrEme Personal Java Virtual Machine (available at http://www.nsicom.com/), and (2) a Palm-OS emulator (Figure 6-c) equipped with a J2ME CLDC Virtual machine.

Starting from the desktop version of Bloof UI, we used AWT2XIML to define the UI transformations for the PocketPC and Palm-OS platforms. Differently from the desktop version, the PDA version of the UI presents: (1) the reorganization of the layout of the central part of the GUI; (2) a tabbed pane to adapts the various parts of the main panel of the GUI over a limited screen, (3) the absence of the tool bar. It is worth noting that without using multiple descriptions of the UI, the developer is obliged to produce different implementations of the UI, each one suitable to the specific platforms that should be used.

Figure 4. Second phase: Multiple GUI’s descriptions process

Currently, the tool permits to manually change the XIML code. We are working on some improvements of the tool. The XIML description will be shown as a graphical tree and the tool allows a developer to visually modify the tree, by adding or removing components, changing its structural properties and so on.

The output of the transformation phase is an XIML file that for each widget contains the transformation rules for different device profiles. This file is joined to the application to produce a deployable archive. This archive is used by TCPTPE server to execute and render the UI on the used device. Differently from existing renderers of XML-based GUIs, the renderer used by TCPTPE is able to dynamically change the presentation of existing GUIs implemented in Java on the basis of transformation rules produced in the previous phase.

To understand the rationale underlying our renderer, a graph-based formalism is used. Each widget is represented as a Components tree (C-Tree) whose nodes can be containers or components and edges represent the relations between containers and contained components. Each node stores the structural properties (e.g. composition, layout, etc.) of both the server side component that it represents and the correspondent client-side component. The separation between abstract GUI components (located on the server) and the concrete device specific components (located on the client) enforces the introduction of two different C-Trees: a server-side C-Tree (SC-Tree) and a client-side C-Tree (CC-Tree).

The mapping between SC-Tree and CC-Tree is performed by the renderer (see Figure 5) following transformation rules when the application instantiates the GUI widget. If no transformation is to be applied, SC-Tree and CC-Tree will be the same.

As shown in Figure 5, these rules are described by an XIML file (a) in the form of a transformation rules’ tree (TRT) (b) and are used to dynamically map the SC-Tree (c) onto a CC-Tree (d). To perform the mapping, each node in the TRT is labeled with a unique ID that binds the node to an SC-Tree component, and stores the structural properties that the related client Half-Object will have on the client.

Thus, given a node n characterized by a unique ID x, the transformation rules used for the generation of the CC-Tree are the following:

(a) If x SC-Tree and x TRT then x CC-Tree
(b) If x SC-Tree and x TRT then x CC-Tree
(c) If x SC-Tree and x TRT then x CC-Tree and a ghost abstract component is created.

In Figure 5, components labeled with “2” and “3” will not be created on the client while the parent of component labeled with “5” will be a new component on the client labeled with “X”. Finally, the component labeled with “4” will be added to the frame.

Figure 5: Third phase: adaptation. Relations among XIML, TRT and C-Trees
of the activities (or tasks) it performs. For each task, one set of Abstract Interactor Objects (AIO) should be defined. Each set represents the abstract graphical components that will be shown in order to run the presentation tasks. Moreover, for each task, a list of requirements that the device must satisfy for the execution should be provided. Starting from the device features used to execute the application, TCPTE will select the tasks that can be executed on device and make the transformation rules for the widgets representing such tasks.

The reverse and adaptation phases can be executed by using a visual tool that guides the developer to define: (1) the tasks; (2) the temporal relations between tasks; (3) the AIOs; (4) the requirements of each task; (5) some other properties related to tasks and AIOs. The tool kit will be released in the next months. Other improvements will be referred to some current implementation limitations of TCPTE, such as the limited number of refactored AWT components.

8. References


