ABSTRACT
The paper describes TCPTE, a framework that supports the development of thin-client applications for mobile devices. By using this framework, Java AWT applications can be executed on a server and their graphical interfaces can be displayed on a remote client. TCPTE combines in a single framework the advantages of thin-client computing with the richness of client-server graphical interfaces and the simplicity of development that characterizes desktop applications.

Categories and Subject Descriptors
D.2.11 [Software Architectures]: Software Architectures – Thin-Client.

General Terms
Design, Languages.

Keywords
Java, Software architectures, Mobile computing, User Interface Adaptation.

1. INTRODUCTION
The widespread of Personal Wireless Devices (PWDs), and their highly distributed, heterogeneous, and mobile nature, have raised new challenges that permeate the entire software engineering life cycle. This new set of challenges is defined as “programming in the small and many”. In fact, key problems of software development for personal wireless devices are related to: (issue 1) the small and many”. In fact, key problems of software cycle. This new set of challenges is defined as “programming in the small and many”. In fact, key problems of software cycle. This new set of challenges is defined as “programming in the small and many”. In fact, key problems of software cycle. This new set of challenges is defined as “programming in the small and many”.

Issue 1 impedes to port most of desktop applications onto PWDs because they typically need high computational power and large memory spaces, that PWDs are not able to provide. For these applications, an interesting approach to execute them on devices with limited resources consists of decomposing the application in groups of components to distribute between a PWD and one or more servers according to the client/server model. In particular, since the considered applications are the interactive ones, the thin-client model represents one of the most interesting architecture to use in the context of mobile computing.

Thin-client architectures belong to two main classes: “distributed presentation” and “remote presentation”.

The first approach, implemented by several protocols and middleware in the last twenty years (for example X Window System and Virtual Network Computing), is characterized by a significant network traffic due to the necessity of communicating keystrokes, mouse movements or bitmap alterations between client and server. This represents a significant obstacle for PWDs due to the limitations imposed by issue 2.

To overcome the problem some researchers [8] propose to refactor 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 GUI components onto different native ones.

Differently from [8] 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 at all issues 3 and 4 and partially consider issue 2.

Issues 3 and 4 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 run-time. The most used approaches \[5, 6, 7\] uses 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. However, the problem of these approaches is to port existing applications from desktop environments to PWDs ones.

Starting from the considerations reported above, we can conclude that in recent years many approaches have been proposed but no particular approach dominates and there is no single dominant wireless-middleware player.

We propose an approach based on the remote presentation schema to overcome most of the problems that characterize the development and the execution of applications for PWDs. The approach, called Thin-Client aPplications for limiTed dEvices \[3\] (TCPTE, available at http://tcpte.rcost.unisannio.it/), is composed of a method to develop or reuse existing applications and a middleware to execute them.

TCPTE supports the development of thin-client Java-AWT applications for PWDs through: (1) the adoption of the thin-client model to reduce computation on client-side (issue 1); (2) an implementation of the thin-client model based on the remote presentation architecture optimized for wireless communication (issue 2); (3) the adaptation of the user interface components to heterogeneous devices thanks to the decoupling of the server-side and client-side portions of the graphical framework through a specific design pattern and a communication protocol (issue 3); (4) the adaptation of the user interface layout through the GUI widgets-transformation based on the device profile (issue 4).

In the following, we will refer to version 1.1 of TCPTE that will be released in the next months. Differently form version 1.0 described in \[3\], many enhancements were added, such as user interface adaptation to limited screen size of PWDs.

2. DEVELOPING AND EXECUTING APPLICATIONS

Our approach to support development and migration of Java AWT applications to client/server environments for execution on PWDs exploits reverse engineering, forward engineering transformations, and incremental rendering of the reengineered GUI onto the target device. 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 completely or partially executed on heterogeneous mobile devices.

TCPTE provides a set of tools to: (1) obtain a GUI meta-model representation of a Java AWT application; (2) edit the GUI meta-model in order to adapt the application GUI to different devices with limited screen sizes; (3) execute the application and migrate it to a client/server environment so that the application logic remains on a server while the user interface is shown on the client according to the features of its graphical system.

The process to execute an application on a PWD by means of TCPTE, either in the case of an existing Java AWT application or in the case of a new one, comprises 4 steps: reverse engineering, packaging, deployment and execution.

2.1 Development/re-engineering and packaging

In the first step (see Figure 1), existing Java AWT applications may be reused or new ones developed and compiled independently of the device on which they will be displayed. Starting from the Java AWT application that we want to use with TCPTE, a tool (AWT2XILM) can be adopted to obtain a meta-description of the application user interface. The language used to describe the GUI is XIML \[1\] that is an XML based language to define user interface models. Among the XML based languages proposed in the literature to define user interface models, XIML was chosen due to its flexibility, expressivities and completeness. However, other languages with the same characteristics could be employed.

![Figure 1. reverse engineering and packaging](image)

AWT2XILM analyzes the structure of a GUI by instantiating the application widgets coded as classes. The creation of a widget produces a tree of components each one labeled with a unique ID. All obtained trees represent the input for the successive step whose aim is to modify the GUI description to enable its rendering on different heterogeneous devices. In this step, a tool helps developer to produce a set of transformation rules that will be used to modify the GUI for each different target platform. The obtained description is an XILM file that represents the transformation rules to use at run-time to show the GUI on the clients. This file is joined to the application to produce a deployable archive (named TCPTE application).

2.2 Deployment and Execution

The archive produced in the previous phase is deployed on the server and the application code will be linked to a specific TCPTE AWT library at run-time. During the deployment, the testing with the new library is performed to verify the need for a re-factoring through the adaptation of the application to the new environment.

Once obtained, the TCPTE application is made available over an IP network by using a TCPTE server. When a remote device requires the execution of the application, its logic will be launched on the server while the graphical components of the GUI will be incrementally shown onto the remote device based on the transformation rules defined for that device.

To transparently support the physical separation of presentation and application logic in a distributed client/server environment, TCPTE provides a java.awt package (in the following called AWTServer) which preserves the same naming and patterns of the native Java AWT but currently offers a limited set of components.
and a few classes of the Swing package to facilitate the migration of more sophisticated GUIs. The implementation of this package interposes a middleware layer between the abstract components of a GUI and its graphical representation, thus moving this on the client device. The division is performed at run time and is transparent to developers and end-users.

Figure 2. Deployment and execution

The whole process of deployment and execution is summarized in Figure 2. In the deployment phase, the application and its XIML description are to be moved on a TCPTE server. Here, the application is dynamically linked to AWTServer. Finally, in the execution phase, using a client for the specific device, the user connects to the server (fig. 2-a) that in turn sends to the client the list of the deployed applications (fig. 2-b). The user selects an application and asks the server for the execution (fig. 2-c). The server creates a new process for the execution of the requested application and its GUI is incrementally shown onto the client (fig. 2-d).

3. TCPTE FRAMEWORK ARCHITECTURE

The TCPTE architecture is shown in Figure 3. The top layer on the server side represents the application that has to be run on the server and whose GUI is displayed onto a remote device.

By using a library substitution approach, the application layer uses the components defined by the AWTServer layer. Starting from a method call to an AWTServer component, the object communication layer allows for invoking the method on the corresponding AWTClient component on the device.

In Java AWT, an application that has to change the GUI state invokes one or more methods on the graphical components, which trigger the invocation of the appropriate methods on the related peers that show the data onto the screen by invoking specific native methods. On the contrary, an input event sent to a GUI graphical component is propagated through the correspondent logical component to the application logic by means of listeners.

In the thin-client architecture, the separation between application and presentation logics raises the necessity of two address spaces in which the two logics can execute independently. By adopting the remote presentation approach, an important task is to individuate the best way to distribute application components between the two address spaces, so to allow the user to interact with the application through a separated device and to leave the application unchanged.

The internal implementation of Java AWT package was modified, and the Half-Object design pattern [2] was used to distribute the graphical components between client and server. By using the Half-Object pattern, each component needs to be split in two objects: one half in the application address space (AAS), the other half in the user interface address space (UIAS). The two half-objects communicate with the protocol offered by the TCPTE Object Communication layer. The details related to AWTServer-Client implementation can be found in [4]

Figure 3. TCPTE Architecture

The AWT layer is also responsible to adapt the application user interface on different client devices based on the device profile. TCPTE uses a GUI renderer that, starting from the GUI instantiated from the application, uses the transformation rules produced in the reverse engineering phase to change the GUI structure and adapt it to the target device. Each GUI 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 Half-Object that it represents and the correspondent client-side Half-Object. 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).

Figure 4. GUI renderer scheme

The mapping between SC-Tree and CC-Tree is performed by the renderer (Figure 4) following transformation rules when the
application instantiates a GUI widget. If no transformation is to be applied, SC-Tree and CC-Tree will be the same.

Below the AWT layer, the Object Communication layer deals with ad hoc serialization/deserialization processes of objects transferred over the network. This layer makes TCPTE independent of the platform as far as the communication mechanisms are concerned. In order to improve the performances of the communication layer, the objects transferred between client and server are serialized in a binary format whereas communication between objects should be based on asynchronous interactions (without blocking the caller) built on a lightweight transport protocol in order to tolerate the high latencies of wireless networks.

As described in [3] the communication between objects should be based on asynchronous interactions built on a lightweight transport protocol to tolerate the high latencies of wireless networks. Starting from these design considerations, TCPTE object communication layer implements a one-way scheme for void method invocations and a synchronous two-way scheme for non-void method invocations, since in this case a specific middleware would be necessary to ensure asynchrony. Unfortunately, a simple substitution of TCPTE RMI middleware with an existing asynchronous one is not immediate when PWDs are used, since restrictions posed by some client platforms prevent these middleware, which are typically designed for desktop computers, to be used on PWDs without significant modifications.

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, the connectivity between client and server could be temporary lost. In TCPTE, when a connection error occurs, the client freezes the GUI and tries to connect to the server. Current release does not tackle session hand-off of TCPTE applications. However, we are working on this improvement.

4. REFERENCES


