# A Remote Display System for Java-based Mobile Applications

Andre Luiz Camargos Tavares
PUC Minas, Institute of Informatics
30535-610 - Belo Horizonte, MG Brazil
andrelct@gmail.com

Marco Tulio Valente
PUC Minas, Institute of Informatics
30535-610 - Belo Horizonte, MG Brazil
mtov@pucminas.br

## ABSTRACT

Remote presentation is an interesting model for executing applications in mobile devices, since applications can be executed on a server and their interfaces displayed on mobile clients. This paper describes a non-invasive and transparent remote presentation system for legacy J2ME applications, called RDA (Remote Display using Aspects). RDA relies on aspect-oriented programming to instrument J2ME applications with remote presentation code. Our first performance results demonstrate that remote presentation is a promising solution for executing CPU-intensive applications in mobile devices or for running applications demanding resources not available in such devices.

## Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications, Client/Server, Mobile Computing Systems*; D.2.11 [**Software Engineering**]: Software Architectures—*Domain-Specific Architectures*

## Keywords

Mobile computing systems; Software architectures; Aspect-oriented programming.

## 1. INTRODUCTION

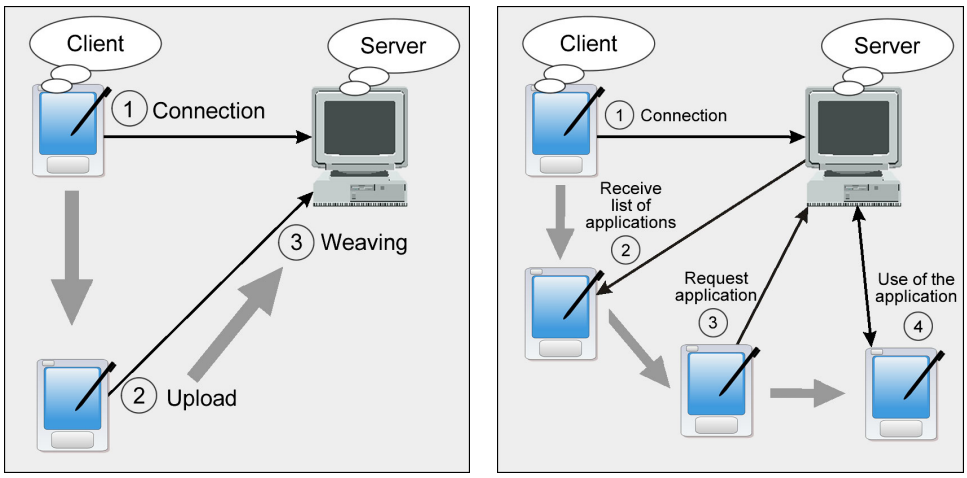
The growing popularity of mobile computing devices, such as personal digital assistants (PDA) and smartphones, presents new challenges to software engineers [8, 11]. When building applications for such devices, developers must consider new problems, including scarce hardware resources, bandwidth fluctuations, voluntary and involuntary disconnections, ad hoc communications, etc [13]. Particularly, limitations in the CPU power and memory capacity currently impact the execution of many applications in mobile devices, including applications that depend on complex algorithms or large amounts of memory. Further, recent improvements in hardware technologies have benefit both workstations and mobile devices, keeping the gap in terms of CPU and memory capacity between the two architectures constant.

For this reason, remote presentation is an interesting model for executing applications in mobile devices. Accordingly to this model, mobile applications should be decomposed in two layers: one to be executed in workstations located in wired networks, containing the application logic and the other containing just the interface of the system to be executed in mobile devices [1, 2, 3, 14]. This decomposition relies on wireless communication just to transmit results and input data between the components in charge of the application logic and interface.

Several solutions have been proposed to support remote presentation over wireless networks. The first kind of solution requires manual decomposition of the target application, in order to insulate application logic and presentation code in different components. An example is the Chroma system [1]. However, such solutions require maintaining two versions of the application: the monolithic version (that executes solely in mobile computers) and the remote presentation version. Other solutions require the use of proprietary frameworks that extend or replace existing GUI frameworks. Examples include ULC [14] and TCPTE [2, 3]. ULC classes resembles the Swing API, but with a ULC prefix (e.g. ULCFrame, ULCLabel, etc). Therefore, the system is invasive and non-transparent to developers. On the other hand, TCPTE reimplements all AWT classes and interfaces, preserving their names. The reimplemented classes contain remote presentation code and also code that implements GUI concerns that must be executed in the server side (for example, processing event listeners or storing information about GUI widgets). For this reason, modifications in the internal logic of the original AWT classes may require modifications in the correspondent TCPTE classes.

This paper describes a remote presentation system for J2ME applications, called RDA (Remote Display using Aspects). RDA offers a *non-invasive* and *transparent* remote presentation system. In other words, legacy J2ME system can be executed remotely, without requiring source code modifications. Moreover, the system takes benefit of aspect-oriented programming to instrument the *original* J2ME Limited Connected User Interface (LCDUI) classes with remote presentation code (i.e. the system does not rely on a library substitution approach). In this way, RDA is robust to changes in the internal logic of the original LCDUI classes. Our first performance results obtained with RDA shows that the speedup gains can be expressive for CPU-bound tasks (such as the Dijkstra shortest path algorithm used in the

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, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
*SAC'08* March 16-20, 2008, Fortaleza, Ceara, Brazil
Copyright 2008 ACM 978-1-59593-753-7/08/0003 ...$5.00.




(a) (b)

**Figure 1: Application deployment (a) and execution (b) in the RDA system**

experiment described in the paper).

The remaining of the paper is organized as follows. Section 2 presents the basic features and the architecture of the RDA system. Section 3 describes the internal aspects proposed by RDA in order to support transparent, non-invasive, remote presentation of J2ME applications. Section 4 presents results about an experiment performed with our prototype implementation of the system. Section 5 discusses related work and Section 6 concludes the paper.

## 2. RDA ARCHITECTURE

**Using RDA:** RDA includes a remote presentation server (a workstation or similar computer) that is responsible for executing J2ME applications (more precisely, for the logic of such applications). In order to benefit from the system, mobile users should first upload to this server the bytecode of the application that will be the target of remote presentation (as described in Figure 1a). Finished the upload of a given J2ME system, RDA automatically starts a weaving process, in order to instrument this system with aspects responsible for redirecting its interface to mobile devices.

In order to execute an application previously transferred to the remote presentation server, a mobile user must first connect to this server, that provides him with a list of available applications (as described in Figure 1b). The user selects the application and requests its execution. From this moment, the application starts running in the server, with its interface being automatically redirected to the mobile device of the user.

**Architecture:** Figure 2 presents the layered architecture of the RDA system. The first layer represents the application that will be executed in the server, with its interface redirected to the client. This layer is executed using a J2ME micro-emulator [4], which emulates in J2SE applications originally developed to J2ME. When the application invokes a method from the J2ME `lcdui` package (package containing classes to implement graphical interfaces), aspects are used to capture these invocations and redirect them to the client. The `lcduiClient` is the component of the system that renders the interface in the client. The serialization layer handles the serialization and deserialization of arguments and results between the server and mobile clients. The communication layer handles the TCP/IP communication between clients and server, using sockets.

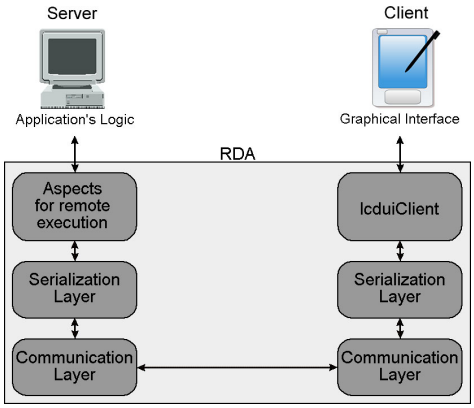



**Figure 2: RDA Architecture**

## 3. REMOTE PRESENTATION ASPECTS

RDA implementation is based on the half-object design pattern [6]. This pattern prescribes dividing an object into two half-objects, one in each address space, with a protocol between them. The idea is to make an object available in two address spaces. In RDA, the half-object pattern is applied over objects associated to GUI widgets, including buttons, lists, text areas, panels, etc. The half-object that resides in the server side of the system is called *server half-object* (SHO); the half-object located in the mobile computing device is called *client half-object* (CHO). Moreover, in the remote presentation server, aspects are used to implement the synchronization protocol between SHOs and CHOs.

When a method or constructor of a GUI component is invoked in the server, aspects intercept the invocation and two threads are started. The first thread continues the normal

execution of the method or constructor in the server, so that the SHO can be updated. However, no information is shown on the server display, because the execution of the method that makes the graphical interface visible is blocked by the system. The second thread serializes the called method and sends the resulting data to the client. Once received by the client, the data is deserialized and the method or constructor is executed in the correspondent CHO. Interactions in the opposite side are triggered by GUI-related events (for example, when the user clicks over a widget). In this case, the CHO serializes and sends the event to its associated SHO, where the correspondent event listener is executed.

We describe next the main aspects that support the synchronization protocol between CHOs and SHOs. The aspects are implemented in AspectJ.

**Aspect that intercept application startup.** The following pointcut implements this interception:

```
pointcut MIDletInicialization(): staticinitialization
    (javax.microedition.midlet.MIDlet+);
```

This pointcut intercepts the static initialization of classes that extend the `MIDlet` class from J2ME[1]. An advice of type `before` associated to this pointcut opens a connection between the server and the mobile device that has requested the execution of the application.

**Aspect that intercept the display of interfaces.** Method `setCurrent` from the `Display` class is the only method responsible for making interfaces visible in J2ME systems. The following pointcut is used to intercept the execution of this method:

```
pointcut setCurrent(Displayable nextDisplayable,
                    Display display):
    call(void Display.setCurrent(Displayable)) &&
    args(nextDisplayable) && target(display);
```

An `around` advice is associated to this pointcut in order to disable the execution of `setCurrent` in the remote presentation server. Therefore, no interface information is displayed on this server. Instead, the advice propagates the call to the `lcduiClient` component.

**Aspects that intercept the instantiation of SHOs.** This interception is implemented over constructors from classes that represent GUI widgets (`Form`, `Button`, `TextField`, `TextBox`, etc). For example, the following pointcut illustrates the interception of the constructor of the class `Form`:

```
pointcut FormConstructor1(String title):
   call(Form.new(String)) && args(title);
```

An `after returning` advice associated to this pointcut captures the SHO created in the server and stores its hashcode in a table. Further, information about the created object is sent to the client, including the class name (`Form`), the constructor name (`FormConstructor1`) and the constructor arguments (`title`). The client then creates its associated CHO.

[1]MIDlet is the name given to J2ME applications that conform to the Mobile Information Device Profile (MIDP), which is the Java runtime environment for smartphones and mainstream PDAs.

There are other 15 similar pointcuts and advices that intercept the creation of the remaining types that can be included in the interface of J2ME applications.

**Aspects that intercept method invocations directed to SHOs:** This interception is applied to methods that change visual properties of SHOs. For example, the following pointcut intercepts the invocation of method `setString` of class `TextField`:

```
pointcut setString(String text, TextField textField):
    call(void TextField.setString(String)) &&
    args(text) && target(textField);
```

An advice of type `before` associated with this pointcut captures the information about the invoked method. Before the method is executed in the server the following information is sent to the client: method class (`TextField`), method name (`setString`), method arguments (`text`) and the hashcode of the target object. If there are arguments that represent SHOs, the server transmits to the client their hashcodes instead. This information is sufficient for invoking the correspondent CHO in the client.

Moreover, the `setString` method is executed normally in the remote presentation server (in another thread in order to turn the communication with the mobile client asynchronous). In this way, methods that retrieve state information about widgets objects (such as `TextField.getLabel`) do not need to be intercepted by the aspect layer and can be processed normally in the remote server.

RDA implementation includes 48 similar pointcuts in order to intercept methods that change visual properties of SHOs.

# 4. EXPERIMENTAL RESULTS

In order to evaluate the performance gains of our prototype implementation of the RDA system in a real setting, an experiment was conducted with a PDA connected to a remote presentation server by an IEEE 802.11, 11 Mbps wireless network. The PDA was a Palm Lifedrive, Intel Xscale 416MHz, 64Mb of RAM, IBM JVM J9 2.2 and the server an AMD Athlon64 1.80GHz, 1024Mb of RAM, Windows XP Professional SP2. The application used in the experiment was the well-known Dijkstra Shortest Path Algorithm, which constitutes the foundation of many location-based applications and services common in handheld computing.

Dijkstra algorithm was executed using graphs with 10, 150, 500 and 1000 nodes. We have also changed the percentage of edges in the graphs to 20%, 30% and 50%, regarding a complete graph. In all the cases, edges were generated randomly in order to obtain a graph with the specified density. The interface of the application was composed by two textboxes to inform the number of edges and the density and a button to request the calculation of the shortest path.

Figure 3 compares the execution time of the Dijsktra algorithm using the RDA system and when the application was fully executed in the PDA (i.e. without remote presentation support). The presented times does not include the generation of the graphs. On the first experiment (Figure 3a, graph with 10 nodes), the result of the RDA system was not favorable, since the small number of nodes in the graph has not demanded enough processing to compensate the communication overhead inherent to using RDA. More precisely, the execution time in the mobile device was very

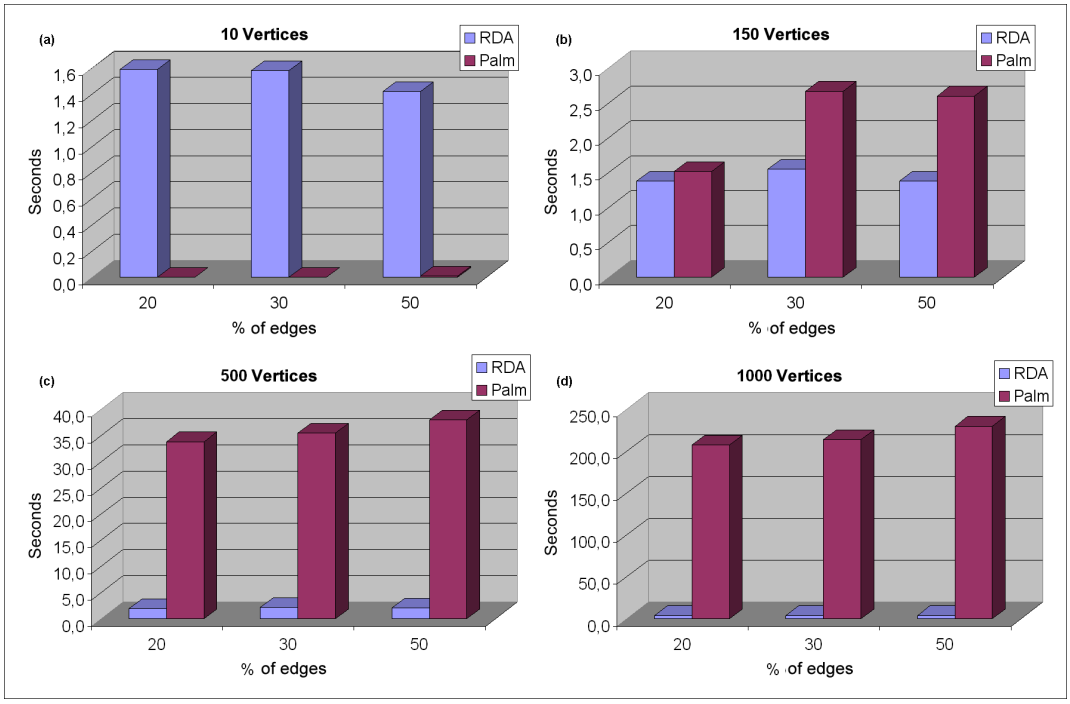



**Figure 3: Results of the Experiment**

close to 0 ms (for 20% and 30% of edges). On the other hand, the execution time using the RDA remote presentation service was superior to 1.5 seconds. Thus, in our networking setting, this value is associated to the latency innate to any round-trip communication using RDA. It is worth to mention that latencies close to 0.5 seconds for one-way interactions have been reported when using other thin-client platforms in Ethernet networks [15].

However, as the number of nodes increases, the efficiency of RDA raises considerably. On the second experiment (Figure 3b, graph with 150 nodes), the execution time using RDA was close to the first experiment, whereas the execution time in the Palm has increased to 2.6 seconds (considering 50% of edges). On the next experiments (Figure 3c and 3d), the execution time using RDA has increased only one second, from one experiment to another. On the other hand, the execution time in the Palm has increased to more than 38 seconds on the third experiment, and further, to more than 230 seconds on the last experiment.

Concluding, the experiment has demonstrated that RDA contributes to reduce considerably the execution time of tasks that demand large amounts of CPU processing or main memory. On the other hand, the system should not be used when the events triggered by the GUI demand low-levels of CPU processing.

## 5. RELATED WORK

Several systems for remote executing applications for mobile devices have already been designed. Among them, we can mention Chroma [1], TCPTE [2, 3] and ULC [14]. In Chroma, domain experts define a group of tactics (or possible alternatives) for decomposing an application, using a so-called remote execution tactics language. In RDA this decomposition is static and transparent to developers: the logic of the system is executed in the remote presentation server and its interface redirected to mobile devices (without the need of source code interventions or the definition of configuration files using particular domain-specific languages).

Thin Client Applications for Limited Devices (TCPTE) [2, 3] is a framework for remote executing Java AWT applications designed for J2SE (while in RDA the objective is the remote execution of J2ME systems). TCPTE architecture for remote presentation is also based on half-objects. However, the system relies on a library substitution approach, requiring the use of its own GUI package, called AWTServer. This package reimplements all original AWT classes (including non-public classes), inserting code related to remote presentation concerns. In other words, remote presentation exhibit a crosscutting behavior in AWTServer, suffering from code spreading and tangling.

In order to customize AWT interfaces to the reduced display of mobile devices, TCPTE supports reverse engineering of interfaces through their transformations to XML-based descriptions [12]. Forward engineering is then used to dynamically render interfaces customized to a given handheld display. However, this solution can not adapt complex interfaces (for example, when an overrich form needs to be divided into more simple forms). Ultra Light Client (ULC) [14] is another remote presentation system that reimplements classes from the Swing API, but with a ULC prefix.

PROSE is an AOP middleware for extending mobile computing applications at run-time [9]. The objective is to proactively adapt mobile applications with functionalities or policies required to their proper execution in a given environment (such as security, logging, coordination with other devices, etc). RDA architecture is similar to PROSE, in

the sense that the system relies on aspects to transparently extend applications. However, RDA supports a single, predefined functionality: remote presentation. Moreover, the system relies on static weaving whereas PROSE is based on dynamic AOP and implements its own weaving engine.

Remote display systems have already been proposed for operating systems. As examples, we can mention the X-Window system [5], common in Unix environments, Microsoft Remote Desktop Protocol (RDP) [7], and Virtual Network Computing (VNC) [10]. However, such systems usually demand significant network bandwidth due to the need to handle low-level interface events, including mouse movements, bitmaps refreshing, text scrollings, and keystrokes. On the contrary, in remote presentation systems network communication is only needed to update the client interface or when an event needs to be processed by the application logic.

# 6. CONCLUSIONS

In this paper, we have described a remote presentation system for handheld computing applications. The proposed system, called RDA, presents the following contributions:

- RDA supports remote execution of legacy J2ME applications, i.e. applications whose interfaces have been originally designed to run on mobile devices. Other systems support standard Java interfaces that usually are not compatible to the reduced size and resolution of handheld displays and for this reason require non-trivial refactoring efforts.
- RDA supports seamless remote execution of J2ME applications, without requiring source code modifications or proprietary frameworks. For this purpose, the system uses half-objects located in presentation servers and in mobile devices. RDA also relies on aspects to implement the synchronization protocol between half-objects. Using aspects, the synchronization protocol was transparently superimposed to the JM2E/LCDUI public classes through the AspectJ weaving process. In this way, the implementation of the system is robust to changes in the internal logic of the original LCDUI classes. Only changes in the public interface of the package require modification in the aspects proposed by the system.

Moreover, our performance evaluation has demonstrated that remote presentation is a promising solution for executing CPU-intensive applications in mobile devices. Remote presentation is also useful when applications demand other resources (such as memory, disk, bandwidth, etc) not available in such devices.

As future work, we have plans to experiment RDA with other mobile systems and wireless technologies (GPRS, Bluetooth, WiMAX, etc). We also have plans to investigate mechanisms to reduce the latency inherent to remote communication, including asynchronous, non-blocking messages, and customizations in the TCP/IP stack (for example, disabling the Nagle Algorithm that prescribes the use of buffers by sender processes).

**Acknowledgment:** This research was supported by a grant from The State of Minas Gerais Research Support Foundation (FAPEMIG).

# 7. REFERENCES

[1] R. K. Balan et al. Tactics-based remote execution for mobile computing. In *1st International Conference on Mobile Systems, Applications and Services*, pages 273–286, 2003.

[2] G. Canfora et al. Developing Java-AWT thin-client applications for limited devices. *IEEE Internet Computing*, 9(5):55–63, 2005.

[3] G. Canfora et al. Developing and executing Java AWT applications on limited devices with TCPTE. In *28th International Conference on Software Engineering (demonstration session)*, pages 787–790, 2006.

[4] J2ME Micro Emulator. http://sourceforge.net /projects/microemulator.

[5] N. Mansfield. *The Joy of X: An Overview of the X Window System*. Addison Wesley, 1993.

[6] G. Meszaros. Pattern: half-object+protocol (HOPP). *Pattern languages of program design*, pages 129–132, 1995.

[7] Microsoft. Remote Desktop Protocol (RDP). http://support.microsoft.com/kb/186607.

[8] H. Ossher, W. Harrison, and P. Tarr. Software engineering tools and environments: a roadmap. In *The Future of Software Engineering*, pages 261–277. ACM Press, 2000.

[9] A. Popovici, A. Frei, and G. Alonso. A proactive middleware platform for mobile computing. In *International Middleware Conference*, volume 2672 of *Lecture Notes in Computer Science*, pages 455–473. Springer, 2003.

[10] T. Richardson et al. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[11] G. C. Roman, G. P. Picco, and A. L. Murphy. Software engineering for mobility: a roadmap. In *The Future of Software Engineering*, pages 241–258. ACM Press, 2000.

[12] G. D. Santo and E. Zimeo. Reversing GUIs to XIML descriptions for the adaptation to heterogeneous devices. In *ACM Symposium on Applied Computing*, pages 1456–1460, 2007.

[13] M. Satyanarayanan. Fundamental challenges in mobile computing. In *ACM Symposium on Principles of Distributed Computing*, pages 1–7, May 1996.

[14] Ultra Light Client (ULC). http://www.canoo.com/ulc.

[15] S. J. Yang et al. The performance of remote display mechanisms for thin-client computing. In *USENIX Annual Technical Conference*, pages 131–146, 2002.