RZ 3306 (#93352) 12/18/00
Computer Science 4 pages

Research Report

Appletizer: Enabling Small Computing Devices as Network Management Consoles

Christian Hörtnagl

IBM Research
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Appletizer: Enabling Small Computing Devices as Network Management Consoles

Christian Hörtnagl

IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland

Abstract

Network and system management tools have benefited from the emergence of Java as an implementation language for applications that are widely portable across platforms in a uniform binary format. The possibility to distribute tasks allows for more scaleable and robust designs [1]. While opportunities for distributed applications following e.g. the mobile agents paradigm are not restricted to Java, this language goes a long way in enabling such applications under the umbrella of a single technology.

For instance, traffic or health monitoring tasks may be placed close to relevant network devices; on the other hand, display tasks belonging to the same network management application may migrate to suit operators. Management scenarios may e.g. require staff attention outside working hours or intervention at remote sites, hence enabling display tasks even on small computing device can improve overall operation in this particular domain.

This extended abstract is concerned with enabling display tasks on a particular class of small devices. It presents a compact mechanism whereby Java programs can display graphical user interfaces (GUIs) inside web pages solely by remote interaction via HTTP and a markup language, such as HTML. Although the approach keeps the normal AWT (Abstract Window Toolkit) programming model and visual appearance largely intact, it does not require a web browser that supports Java applets.

Owing to the portability of Java byte code and an integral class library, the Java programming language has become a preferred choice for implementing graphical front-ends in multi-tiered applications. The possibility to embed Java applets inside markup pages has helped to push this trend, because Internet connectivity via the World Wide Web is becoming widespread. By off-loading work from central application servers on demand, Java-enabled clients, such as mainstream web browsers, promise richer and more intuitive graphical user interfaces (GUIs), shorter response times and better scaleability for distributed applications.

On-demand migration of Java code is particularly appropriate for network and system management tasks, because a large number of potential configurations (e.g. in terms of technologies or problem cases) maps into only few actual ones at each time. In the context of this work, we ignore this aspect however, and concentrate on Java-based GUIs instead.

Java Virtual Machines (JVMs) are now widely deployed on workstations and PCs, in particular since they are bundled with web browsers for popular desktop operating systems. At present this is not yet the case for many small computing devices such as Personal Digital Assistants (PDAs), whose web browsers, where available, remain of a more restricted (Java-less) sort. The upcoming generation of cellular phones with built-in support for the Wireless Application Protocol (WAP) and its XML dialect WML offers some client-side programmability, albeit of a more limited and also Java-less variety. Sun has translated known constraints of electronic consumer products into a set of Java specifications with reduced requirements, and major industry players work on putting JVMs on new generation of cellular phones [2], but wide-scale deployment following these trends has not yet occurred.

There exists a class of small computing devices on the market for which JVMs are not available, either because of lack of porting effort or because their resources are too constraining. Some of their common features such as simple interfaces and TCP/IP network connectivity, small form factor, and potential appeal to a large consumer audience make them attractive as consoles for access to applications hosted inside the global Internet. PDAs with the Windows CE operating system (e.g. from Philips) and high-grade cellular phones (e.g. from Nokia) belong to this category. Small computing devices have the potential to enhance application usage both in terms of function, e.g., system administrators may receive alarm notifications while away from their offices, and in terms of extent, e.g., consumers may seamlessly tap into information stored on the Internet regardless of their location and present setting.

While advancements in technology undoubtedly will soon remove specific resource constraints, market considerations will continue to give rise to entry-class models for which a remedy similar to the presented one remains valid in order to uphold their competitive position.

In this extended abstract we present a simple mechanism whereby Java programs can display GUIs inside web pages solely by remote interaction via HTTP and a markup language, such as HTML. Although our approach keeps the normal AWT (Abstract Window Toolkit) programming model and visual appearance largely intact, it does not require a web browser that supports Java applets.

The service is achieved in a prototype implementation by introducing a special web server that acts as an intermediary between a Java applet and a web browser. It has the capability to render calls to the Java AWT class library for presenting graphical user interfaces (GUIs) into HTML expressions. The basic architecture is depicted on the left part of Fig. 1.

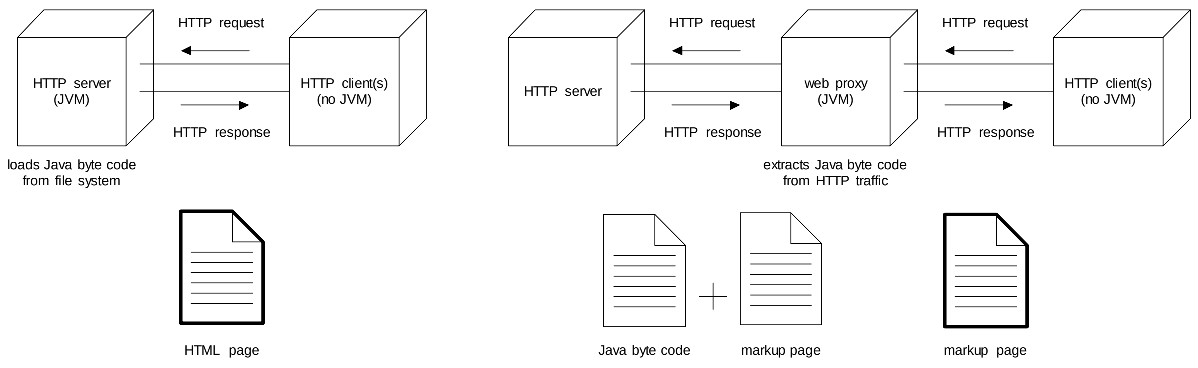

Fig. 1: prototype configuration (left) and final deployment scenario (right).

It benefits from the fact that Java AWT class library and HTML (or similar markup) possess similar expressive power for describing graphical user interfaces. In particular, they make use of a set of similar graphical components (widgets) that comprise dialog boxes and web pages respectively. Hence an automatic conversion can be achieved for a relevant subset of expressions.

We have developed and tried this solution with tools from the Distributed Management Framework (DMF) [1] with the specific intent of keeping the original code base intact. Fig. 2 shows a simple example of such transparent transcoding.

We are focusing on Java applets, because for many Java applications (not applets) the AWT class library is now considered outphased and the Swing library is used instead. Although many of its core classes directly resemble older AWT classes, it would seem that there is a fundamental problem with mapping Swing to HTML in particular, because the design of Swing allows for an open set of graphical components. A general solution consists of rendering Swing library calls into bitmap images and embedding these inside HTML as clickable images

The prototype functions as a web server. For final deployment, we foresee a markup transcoder that is configured as a transparent web proxy (right part of Fig. 1); it inspects passing HTML (or XML) pages for *APPLET* tags. Applet execution and all associated state are kept on the web proxy (instead of the client); output from the GUI rendering process is merged into the remaining markup description that also arrives from the server and is pushed onto the (JVM-less) clients. On the upstream receiving side the proxy deals with a combination of markup descriptions (e.g. HTML with *APPLET* tags) and related Java byte code; on its downstream sending side it forwards enriched markup descriptions (without *APPLET* tags). In addition to performing the necessary markup transcoding and user feedback processing, the proxy also hosts the execution of the intercepted Java applets. A proxy that is also implemented in Java (as is the case for the prototype) could easily migrate to convenient locations inside the network, e.g. to the perimeter of a wireless portion of the network. Furthermore, the proxy could e.g. be informed about smaller screen sizes, etc., by policies that instruct and refine its transcoding decisions. We notice in passing that a similar solution may also be attractive for security reasons, when executable code must not be downloaded all the way into clients.

With this automatic conversion in place the dynamic description of how GUIs interact follows conventions that are thoroughly familiar to Java programmers and that fit into the surrounding software infrastructure. This scheme is clearly more flexible than retrieving static

HTML pages and more supportable than ad-hoc compilations of application data into HTML as is normally done in markup-embedded or CGI scripts.

Overall, we aim at a solution that does not violate any of the programmatic interfaces that were put in place by AWT. In particular, the targeted applications will request core classes from the package `java.awt` (or an exact look-alike), and our implementation has to enforce modified semantics on the methods so accessed. By maintaining compatibility (or upward compatibility) to AWT, clients can treat the actual type of display as largely opaque information (when screen sizes permit), and a single code base can suffice for deployment scenarios including e.g. both Java-enabled and restricted, "first-generation" web browsers as clients.

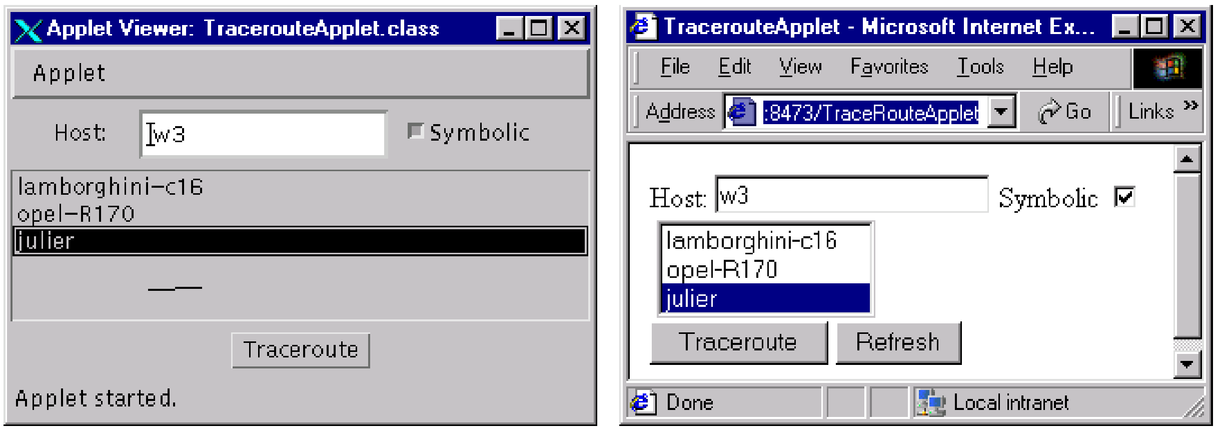

Fig. 2: Java applet as front-end to traceroute utility; left running inside web browser, right automatically transcoded into markup.

A particular challenge involves dealing with asynchronous feedback from clients: HTTP is a specific protocol instance for client-server interaction, where clients expect responses exclusively after previous requests of their own. Only a rudimentary form of server-initiated interaction is foreseen: it cannot be driven by asynchronous events, but requires a-priori knowledge of fixed delay intervals. This strict asymmetry between clients and servers does not match how AWT assigns roles: user interface updates (HTTP responses) normally result from user feedback (HTTP requests), but they may also occur under program control at any later time. This is, e.g., typically the case when default values need updating (HTTP responses) to final results (no previous HTTP request in synchronous relationship) after long background calculations. We foresee different ways to overcome this discrepancy and have implemented one suitable solution in the current prototype (transcoder adds "Refresh" button to the markup on-the-fly).

Approaches that are partly similar to what we address have been described before e.g. in relation to WAP/WML, GSM/SMS or Palmpilot-specific output formats [3, 4]. Ours is specific because of its inherent potential for supporting transparent reuse of existing Java applets in new deployment scenarios (involving Java-less clients). As such, it is also more ambitious than these alternatives.

In summary, we have described a mechanism whereby Java programs that form GUIs by the normal conventions of the Java AWT class library can make use of small devices with primitive web browsers, but without requiring JVMs. Some members of the current generation of PDAs and cellular phones qualify as examples. Both from the points of views of the programmer and the user the programming interface and the visual experience of the offered interaction respectively stay conveniently close to what is normally experienced on desktop installations and delivered by existing applications. Our solution provides a possible low-cost path for deploying Java GUIs on small devices for structured data display and for obtaining user feedback.

We have experimented with this approach with samples from an existing code base of network management applications, with the specific intent of making display tools accessible also on small computing devices and without having to rewrite any significant amounts of source code. We have found that the presented solution fulfills these requirements well.

References

[1] M. Feridun, W. Kasteleijn and J. Krause. Distributed management with mobile components. in Proc. *6th IFIP/IEEE International Symposium on Integrated Network Management* (IM 99), 1999.

[2] Symbian. *EPOC's Java Implementation*. EPOC White Paper, March 1999.

[3] T. F. La Porta, R. Ramjee, T. Woo and K. K. Sabnani, Experiences with network-based user agents for mobile computing, *ACM/Balther Journal of Mobile Networks and Applications*, 3(2), August 1998.

[4] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, Adapting to network and client variation using infrastructural proxies, *IEEE Personal Communications Magazine*, 5(4), August 1998.