Mini: A Minimal Platform Comparable to Jini for Ubiquitous Computing

Polly Huang, Vincent Lenders, Philipp Minnig, and Mario Widmer

ETH Zürich, Zürich ZH 8092, Switzerland,
{huang, lenders}@tik.ee.ethz.ch,
{pminnig, marwidme}@ee.ethz.ch

1 Introduction

Ubiquitous devices, for example mini-laptops, PDAs, mobile phones, and home appliances, have the potential of accomplishing collaborative tasks and bringing convenience to life. The challenge of realizing such a network of collaborative devices lies in how we handle the inherited heterogeneity, mobility, and limited resource problems. These devices can have very distinct hardware and software profiles, thus heterogeneous. They can move very dynamically among several networks, thus mobile. They can have very limited computation capability and memory space, thus resource constrained. System software for these devices must be able to deal with these problems.

Java and Jini [9] are among the most prominent solutions to overcome the heterogeneity problem and allow service discovery for dynamic devices. This service discovery functionality is particularly important in this ubiquitous network context where the devices are migrating and services are often specific to the local environment. It is difficult to anticipate the kind of services that will be needed. Besides, these devices are limited in memory space. It is not practical to install all probable services all at once. We think, in this context, a minimum system software stack that supports dynamic service discovery and loading is much more appropriate.

Taking the hands-on approach, we have selected Java and Jini as the reference system. We have shown [1] that existing Java and Jini implementations, although supporting cross-platform and seamless service discovery, are too large in code size for the devices under consideration here. Following up that finding, we set out in this work with the objective of implementing a minimal platform comparable to Jini so the stack fits in the limited flash memory space. We think reducing system software size is critical to the success of a true ubiquitous network, i.e., devices everywhere. The reason is that the smaller the system software stack, the lower the manufacturing cost and the wider range of devices to be included in the collaborating mass – therefore increases the coverage and the impact of the ubiquitous devices.

Having identified the size bottleneck of Java/Jini stack being the use of RMI and unnecessary functionalities in the context of ubiquitous devices, we 1) propose to exchange service description instead of service proxy between the lookup

server (LUS) and other Jini entities, which allows us to decouple the use of RMI easily, and 2) implement only the minimum service discovery functionality. This simplified form of Jini is referred to as Mini.

Using Compaq iPAQs as our experimental hardware platform, we have achieved in implementing Mini with similar Jini API and reduced the Mini/Java stack size to a mere 1'133 KB. That is approximately a factor of 20 smaller than Sun Microsystem's implementation of Jini/Java and the size of Mini itself is approximately a factor of 4 smaller than the smallest Jini implementation we have come to know.

2 Related Work

There are numerous efforts from the research and the commercial communities to extend the use of Jini to ubiquitous (or resource limited) devices. Sun Microsystems introduced the Surrogate Architecture [4]. In the Surrogate Architecture, surrogate hosts act as Jini proxies for the limited devices. These limited devices can be attached to the proxy by any proprietary protocol. Although, the Surrogate Architecture enables limited devices to be part of the Jini concept, it radically reduces the degree of spontaneity offered by Jini.

Attempts to improve the size and performance of RMI were realized by [10]. Others [2, 5], tried to avoid the RMI layer for Jini. They re-implemented Jini with a substitute for RMI. These solutions focus on alternative communication models to replace RMI. Our solution takes another step further. In addition to removing Jini's dependency on RMI, we include only the necessary service discovery functionality in our platform.

Yet others optimized the Java 2-RMI stack. SavaJe [7] is a Java Operating System for embedded devices that fits into 12 MB ROM. Tini [8] is a hardware solution supporting a subset of the Java 1.1 platform in a small footprint of approximately 50 KB. However, serialization and reflection is not supported. PsiNaptic [6] developed a proprietary Jini implementation with a footprint of approximately 100 KB running on the TINI platform. These solutions are proprietary and some hardware specific. Our solution with comparable code size is open source, software based, and inter-operable with the above solutions.

3 Approach

Earlier research [1] to run Jini on limited devices showed that RMI is a performance and memory bottleneck in the Jini stack. Furthermore, Jini uses the Java 2-RMI interface which is not compatible with the reduced Java platforms as PersonalJava, EmbeddedJava, or even not present as the case of Java 2 Micro Edition (J2ME). Remote object activation, object marshalling, and the Java 2 security scheme have been introduced first since Java 1.2. Class reflection and the class loader mechanisms have undergone changes from Java 1.1 (PersonalJava and EmbeddedJava).

Instead of adding these lacking mechanisms, providing a slightly better performance and system software size, Mini was implemented from scratch. Our approach focuses on the requirements of ubiquitous device computing while avoiding RMI for remote invocation of distributed objects, thus, reducing the computational and memory resource requirements of the software stack.

The idea was to keep the basic key concepts of Jini in Mini. Mini provides the following features:

- Clients and services are able to discover lookup services without a priori knowledge.
- Services are able to register with lookup services.
- Clients and services are able to renew registrations via leases.
- Clients are able to discover and request services for usage.
- Lookup services return a list of registered services upon demand.
- Lookup services notify all entities of expired services.

Furthermore, the implementation of Mini is categorized in the following way:

- Mini runs in heterogeneous environments and on resource limited devices. A full Java 2 Virtual Machine (VM) is not required by Mini but instead a Virtual Machine conform to JDK1.1.8 as EmbeddedJava or PersonalJava. In this work, Kaffe [3] was used.
- Mini is implemented without the RMI classes contained in the `java.rmi` package.
- Mini consumes as little resources (memory and CPU) as possible.
- The Mini API is similar to Jini and accessed intuitively by Jini programmers.

4 Evaluation

In this section, a quantitative and qualitative comparison with Jini highlights the achieved results.

The total ROM size of the Mini classes is 26 KB. This size is comparable to the core classes of Jini contained in `jini-core.jar` and `reggie.jar`. These two packages together require 258 KB. This is a reduction by the factor of ten. Obviously, this size improvement also implicates lacking features in Mini. These lacking features are highlighted later in this section.

Sun's Jini (v 1.1) implementation running on JDK 1.3.1 with RMI was used as reference. The Java 2 core classes consume 17 MB of ROM space (this value may vary for different Java 2 environments). With the Virtual Machine, this configuration requires a total amount of 17'758 KB. In contrast, Mini running on Kaffe consumes a total memory of 1'133 KB which is a reduction of the stack of more than a factor of 10.

The RAM requirements of Mini were elicited by measuring the heap size of the Mini LUS without any registered services. The heap size oscillated between 250 KB and 900 KB. Furthermore, measurements of the RAM consumption at

execution showed that approximately 80 percent of total memory (heap and VM) can be saved when running Mini on Kaffe compared to Sun's JDK.

Qualitatively, the optimized implementation of Mini differs from the Jini paradigm in the following aspects:

- Mini lookup servers only store ServiceDescription objects of registered services unlike proxy objects in Jini. Services do not upload their proxy classes or objects to lookup servers but make them available for download at an HTTP server for clients. This way, service registration entries require less memory at LUS and the resource consumptions are better distributed among the entities.
- Mini lookup servers do not have to register objects with the remote method invocation daemon (rmid). The Mini lookup server is a stand-alone application. On the other hand, remote activation and deactivation of the lookup server is not possible with Mini.
- In Mini, the service interface must not be a priori known by the client. There is no need for well-known interfaces. Nevertheless, service objects can be downloaded dynamically by clients. The client uses class reflection to determine the methods provided by the service.
- Leases are handled differently in the Mini implementation. Lease times are specified by the requester in contrast to the lease grantor in Jini. In addition, lookup services do not remove service registrations of unused services unless the service did not renew its lease.
- Mini does not identify services with unique IDs. Services are uniquely identified with ServiceDescription containing a name associated with a network address.

5 Summary and Outlook

Mini is a minimal platform for collaborating, heterogeneous devices. It provides the minimal features for robust and dynamic service discovery in the context of ubiquitous computing. With Mini, we achieve of building a system similar to Jini with the size reduced by a factor of 10 and even by a factor of 20 for the total Java/Jini stack which is comparable to our Java/Mini stack. The former size reduction is primarily contributed by implementing only the necessary service discovery functionality. Replacing service proxies in Jini with service descriptions in Mini also helps, but the major effect of this result is the decoupling of RMI for which we are able to do away without a full-blown Java VM. That is the reason that we are able to use Mini with a variety of Java VMs without specific RMI support. This further contributes to the size reduction for the combined Java/Mini stack with a factor of 20. The stack is now 1'133 KB large in size and fits in all systems with 2 MB or larger ROM space which covers all the PDAs and most embedded system development boards we come to know.

There are two potential directions of future work. One is the interoperability issue between Mini and Jini that we inherit by changing the communication

model. A translator gateway can be used to facilitate such inter-operation. Furthermore, Jini, and Mini so far, assumes at least one lookup server per subnet and discovers services within the subnet. In a multi-hop device network consisting of several subnets, the lookup servers will not be exchanging their service registries across subnets so devices on one subnet will not have a chance of discovering services provided elsewhere. To overcome this deficiency, we currently adopt a solution similar to Jini's where all lookup servers within a community are known a priori so local service registries and event notifications can be exchanged among all lookup servers on different subnets.

References

1. Lenders V., Huang P. and Muheim M.: Hybrid Jini for Limited Devices. In Proceedings of the IEEE International Conference on Wireless LANs and Home Networks, ICWLHN, Singapore, pp. 27-34, 2001.
2. Deter S. and Sohr K.: Pini - A Jini-like Plug & Play Technology for the KVM/CLDC. Innovative Internet Computing Systems, LNCS 2060, pp. 53-67, Springer, 2001
3. Kaffe: website. http://www.kaffe.org, 2002.
4. Sun Microsystems: Jini Technology Surrogate Architecture Specification. http://surrogate.jini.org/sa.pdf, Version 1.0, 2001.
5. Preuss S.: Netobjects - Dynamische Proxy-Architektur für Jini. In Proceedings of Net.ObjectDays2000, pp.146-155, Net.ObjectDays-Forum c/o tranSIT GmbH, 2000.
6. PsiNaptic: website. http://www.psinaptic.com, 2002.
7. SavaJe: website. http://www.savaje.com, 2002.
8. Loomis D.: The TINI Specification and Developer's Guide. Addison-Wesley, NJ, 2001.
9. Sun Microsystems: Jini Technology Architectural Overview. http://www.sun.com/jini/whitepapers/architecture.pdf, 1999.
10. Java Community Process: RMI Optional Package Specification. Version 1.0, JSR 66, 2002.