Adaptable Mobile Applications through SATIN: Exploiting Logical Mobility in Mobile Computing Middleware

Stefanos Zachariadis and Cecilia Mascolo
Dept. of Computer Science, University College London
Gower Street, London WC1E 6BT, UK
{s.zachariadis,c.mascolo}@cs.ucl.ac.uk

1 Introduction

With the recent developments in wireless networks (Wavelan, Bluetooth) and the sales of mobile computers of any kind (such as laptop computers, Personal Digital Assistants (PDAs), mobile phones etc.) soaring, we are experiencing the availability of increasingly powerful mobile computing environments that can roam across different types of networks. We have also recently witnessed the acceptance of Logical Mobility (LM) techniques, or the ability to ship part of an application or even a complete process from one host to another. The increasing popularity of the Java programming language and environment has largely been correlated with the acceptance of logical mobility techniques, due to the inherent code mobility infrastructure that Java provides. As such, LM techniques have been successfully used to enhance a user's experience (Java Applets), to dynamically update an application (Anti-Virus software etc.), to utilise remote objects (RMI, Corba, etc), to distribute expensive computations (Distributed.net) etc. Whereas various mobile middleware systems have been developed, the use of logical mobility in those systems has been very limited. The purpose of this work is to investigate the use of logical mobility in mobile computing environments, our hypothesis being that it can bring tangible benefits to both application engineers and users which cannot be supported by the current state of the art. In particular, we wish to tackle software, hardware and application heterogeneity as well as environment dynamicity (service discovery and advertising) in mobile environments, using LM techniques. The use of LM techniques would also allow for more efficient use of peer and local resources and overall would advance us further towards ubiquitous computing and self-healing applications, by allowing for a greater degree of adaptability, dynamicity and reaction to context.

Current research into this area can be roughly split into two categories: Approaches which use LM to provide reconfigurability in the mobile computing middleware itself, allowing applications to interact with services provided by heterogeneous platforms and middleware systems and approaches that use particular paradigms of LM to provide a particular form of functionality to applications. Examples of the first category include ReMMoC[2] and UIC[7]. Examples of the second category include Lime[6], PeerWare[4] and Jini[1]. The main problem we find with these approaches is that they are not flexible enough: Despite offering innovative solutions, their use of LM is limited to solving problems of a limited scope.

2 Logical Mobility: Paradigms, Abstractions and Classifications

Logical Mobility is defined as the ability to send parts of an application (or a complete application) to other hosts on a network. Using abstractions defined by modern programming languages, we can define the following items that can be transferred across the network: Classes, Objects, Remote Procedure Calls (RPC) and Application Data. Note that in this context, application data may include code that cannot be directly mapped to the underlying platform (for example, sending Python code to a Java runtime). We define a Logical Mobility Unit (LMU) as a combination of any of the above. The ability to move parts of an application around a network has already been classified into paradigms (details in [5]). However, in a mobile computing scenario, we add an extra dimension to those paradigms, the communication semantics. In a traditional distributed system (DS) synchronous communications is the most common communication paradigm, given the reliability of the network connection. In a mobile distributed system however, we are encountering both synchronous and asynchronous communications, with the latter being the norm, as mobile connectivity is usually fluctuating and error-prone. When tackling LM, there are more considerations that we need to take into account: How to use the received LMU, how to identify an LMU so that we are able to request it, how to formulate and structure an LMU for transfer and how to verify that the target platform is able to execute the received LMU, a problem arising from hardware and software heterogeneity.

3 SATIN: Engineering Self-healing mobile applications

In previous work[8, 3], we have identified a number of examples showing that LM can bring tangible benefits to mobile applications and have identified a number of requirements needed to harness this technology in a mobile environment. As such, we present SATIN,

a low footprint component based middleware for mobile distributed systems.

The main component of SATIN is the *core*, a container where various components, or *capabilities* in SATIN terminology, of the middleware are registered to. A capability can be anything from a library providing some functionality, such as TCP/IP connectivity or compression, to an application (SATIN applications are a collection of capabilities). Capabilities are identified to the system using a string identifier. We require that no two different capabilities with the same identifier are registered to the same core and recommend a unique identification scheme, similar to that employed by PalmOS applications. SATIN also provides a versioning and dependency scheme for capabilities using the capability identifier. Different capability archetypes are defined: we focus on Advertised Capability.

SATIN tackles discovery and advertising services as capabilities. This allows for different mechanisms to be used according to the environment: Multicast, publish/subscribe systems, interoperability with other systems, etc are all allowed. SATIN allows capabilities to be flagged as *advertisable*, which allows them to express information which is of interest to other hosts in an XML advertising protocol we have defined. For example, a file transfer capability may signal that it is listening on port 21. A SATIN advertiser capability can then use this message to advertise it. Advertised capabilities can select which capabilities to allow to advertise them, by using the advertiser's capability identifier. The advantage of using identifiers which are globally registered, is that it allows advertising a capability without transmitting its stub/interface over the network, as it can be assumed that clients of the capability will be aware of its interface. However, an advertiser can potentially advertise a stub or interface as well. Advertising capabilities can be advertised themselves, so that for example peers can be informed that a particular multicast advertising group is currently in reach.

Using the techniques mentioned above for discovery and advertising, SATIN has the ability to dynamically send and receive LMUs, through the use of the Logical Mobility Deployment Capability (LMDC). An LMU can encapsulate any type and paradigm of LM, as defined in Section 3, including complete capabilities. The functionality of the LMDC is directly exposed to SATIN applications. SATIN is completely modular, the only required component being the core. It can be statically configured, meaning that no new capabilities can be registered, or dynamically configured, allowing for the registration of new capabilities at runtime.

To develop an application for SATIN, it is pivotal to try and decompose it into separate capabilities, as this promotes code-sharing and interoperability between applications running on the same host (or multiple ones), as well as ease of software maintenance. Components of an application which contain functionality which may be reused (a compression technique for example), should be encapsulated in a capability. SATIN also provides applications with a range of capabilities, ranging from XML parsers to communication paradigms, which should be used if possible. If not, alternative functionality should be provided in the form of a capability.

The current prototype of SATIN, implemented in Java, weighs in at around 64KB. We have implemented a dynamic program launcher[9] on top of SATIN, that updates all capabilities installed or installs new ones transparently, using any peers in any network that are within the current reach. We plan to evaluate our approach further, by testing applications ranging from application-level Mobile Agent routing, to adaptability to different services offered by different middleware platforms.

4 Conclusion

With respect the approaches mentioned we provide an *all-purpose* mobile computing middleware, which is flexible enough to directly provide the ability to use any of the LM paradigms, without imposing any constraints on how to use them. Our middleware can be used to build capabilities that work like (or interoperate with) the approaches mentioned, but its usefulness is not limited to just this type of functionality. SATIN provides a single layer which tackles the problem inherent to mobility, i.e., heterogeneity. To substantiate our claims, we have designed a prototype middleware, SATIN, which implements our principles. We are currently in the process of testing our work with various applications, as to prove that this added flexibility comes with reasonable performance trade-offs. Initial results show that SATIN and a sample application takes 1155KB of heap memory and it takes 1452ms to install a capability remotely.
Acknowledgements This work is being sponsored by EPSRC grant number GR/R70460.

References

[1] K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. *The Jini[tm] Specification*. Addison-Wesley, 1999.

[2] L. Capra, G. S. Blair, C. Mascolo, W. Emmerich, and P. Grace. Exploiting reflection in mobile computing middleware. *ACM SIGMOBILE Mobile Computing and Communications Review*, 1(2), 2003.

[3] L. Capra, S. Zachariadis, C. Mascolo, and W. Emmerich. Towards a mobile computing middleware: a synergy of reflection and mobile code techniques. In *Proceedings of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems*, 2001.

[4] G. Cugola and G. Picco. Peer-to-peer for collaborative applications. In *Proceedings of the IEEE International Workshop on Mobile Teamwork Support, Collocated with ICDCS'02*, July 2002.

[5] A. Fuggetta, G.P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Trans. on Software Engineering*, 24(5).

[6] G.P. Picco, A. Murphy, and G.-C. Roman. LIME: Linda meets Mobility. In *Proc. 21st Int. Conf. on Software Engineering (ICSE-99)*, pages 368–377. ACM Press, May 1999.

[7] M. Roman, F. Kon, and R. H. Campbell. Reflective middleware: From your desk to your hand. *IEEE Distributed Systems Online Journal, Special Issue on Reflective Middleware*, July 2001.

[8] S. Zachariadis, C. Mascolo, and W. Emmerich. Exploiting logical mobility in mobile computing middleware. In *Proceedings of the IEEE International Workshop on Mobile Teamwork Support, Collocated with ICDCS'02*, July 2002.

[9] S. Zachariadis, C. Mascolo, and W. Emmerich. Adaptable mobile applications: Exploiting logical mobility in mobile computing. In *5th Int. Workshop on Mobile Agents for Telecommunication Applications (MATA03)*. LNCS, Springer, To appear.