Exploiting Reflection and Metadata to build Mobile Computing Middleware

Licia Capra*, Wolfgang Emmerich and Cecilia Mascolo

Dept. of Computer Science
University College London
Gower Street, London, WC1E 6BT, UK
{L.Capra|W.Emmerich|C.Mascolo}@cs.ucl.ac.uk

1 Introduction

Recent advances in wireless networking technologies and the growing success of mobile computing devices, such as laptop computers, third generation mobile phones, personal digital assistants, watches and the like, are enabling new classes of applications that present challenging problems to designers. Devices face temporary and unannounced loss of network connectivity when they move; they discover other hosts in an ad-hoc manner; they are likely to have scarce resources, such as low battery power, slow CPU speed and little memory that need to be exploited efficiently; they are required to react to frequent changes in the environment, such as new location, high variability of network bandwidth, etc.

To help designers building mobile applications, middleware that faces the problems arisen by mobility should be put in place. In the past decade, middleware technologies [6] built on top of network operating systems have greatly enhanced the design and implementation of distributed applications. In particular, they succeeded in hiding away many requirements introduced by distribution, such as heterogeneity, fault tolerance, resource sharing, and the like, from application developers, offering them an image of the distributed system as a single integrated computing facility [5]. These technologies have been designed and are successfully used for stationary distributed systems built with fixed networks, but they do not appear suitable for the mobile setting [1]. Firstly, the interaction primitives, such as distributed transactions, object requests or remote procedure calls, assume a high-bandwidth connection of the components, as well as their constant availability. In mobile systems, in contrast, unreachability and low bandwidth are the norm rather than an exception. Secondly, object-oriented middleware systems, such as CORBA, mainly support synchronous point-to-point communication with at-most-once semantics, while in a mobile environment it is often the case that client and server hosts are not connected at the same time. Finally, traditional distributed systems assume a stationary execution environment that contrasts with the new extremely dynamic scenarios. While it was reasonable to hide completely context information (e.g. location) and implementation details from the application, it now becomes more difficult and makes little sense. By providing transparency, the middleware must take decisions on behalf of the application. The application, however, can normally make more efficient and better quality decisions based on application-specific information.

In this paper, we first analyze the requirements that middleware for mobile computing should meet. We then describe a mobile middleware based on the principles of reflection [4] and metadata [3]. We briefly illustrate a first prototype we have developed and then conclude the paper by pointing out future directions of research.

* Contact Author: Licia Capra. Phone: +44 20 7679 3645. Fax: +44 20 7387 1397.

2 Requirements of Mobile Computing Middleware

Middleware for mobile computing needs to be *light-weight*; it should support an *asynchronous* communication paradigm between components and should allow applications to be *aware* of their execution context, for the reasons we discuss below:

Light Computational Load. Mobile applications run on resource-scarce devices, with little amount of memory, slow CPUs, limited battery power, etc. Running high-performance but heavy-weight middleware systems on these devices is not feasible, because of resource limitations. It is therefore necessary to trade-off between computational load and non-functional requirements achieved by the middleware. This might mean, for example, to relax the assumption to keep distributed data replicas always tightly synchronized, and allow the existence of diverging replicas that will eventually be reconciled.

Asynchronous Communication. Mobile devices connect to the network opportunistically for short periods of time, mainly to access some data or to request a service. Even during these periods, the available bandwidth is by order of magnitude lower than in fixed distributed systems and it may suddenly drop to zero if an area without network coverage is entered. It is often the case that the client asking for a service, and the server delivering that service, are not connected at the same time. In order to allow interaction between components that are not executing along the same time line, an asynchronous form of communication is necessary. For example, it might be possible for a client to ask for a service, disconnect from the network, and collect the result of the request at some point later when able to reconnect.

Awareness. Mobile systems execute in an extremely dynamic context. Bandwidth may not be stable, services that are available in a particular moment may not be there a second later, because, while moving with our hand-held device, we may change location and loose connection with the service provider, etc. As we cannot foresee all possible execution contexts of applications, there is no static knowledge developers can transfer to the middleware to enable it to decide transparently how to behave in different situations. To avoid poor performance and unusability, applications need to interact with the underlying middleware, in order to become aware of their execution context and dynamically tune middleware behaviour accordingly. For example, middleware cannot transparently decide on which hosts to create replicas of a bunch of data independently of the application, because those hosts may not be there any longer when the application needs to access that information. A better choice would be to enable the application to instruct the middleware on the hosts it should employ as replica-holders, maybe using the host on which the application is sitting, in case the application knows it is going to disconnect from the network for a considerable period of time (for example, because the battery is low).

3 Reflection and Metadata

As pointed out in the previous section, a basic requirement for mobile computing middleware is context awareness. Middleware must interact with the underlying network operating system and keep updated information about the execution context in its internal data structures. This information has to be made available to the applications, so that they can listen to changes in the context (i.e., *inspection* of the middleware), and influence the behaviour of the middleware accordingly (i.e., *adaptation* of the middleware).

A key issue in mobile computing middleware research is therefore how to enable this bi-directional flow of information and two-way interaction between middleware and applications. We believe reflection and metadata can be used to build middleware systems that provide the context-awareness required for mobile computing.

Metadata. Through metadata we obtain separation of concerns, that is, we distinguish what the middleware does from how the middleware does it. In particular, each application encodes in an *application profile* (i.e., in the middleware metadata) meta-information regarding *how* the middleware has to behave *when* executing in particular contexts. More precisely, a profile contains two types of information:

- passive information, where the application asks the middleware to listen to changes in the execution context and to react accordingly, independently of the task the application is performing at the moment. For example, the application may ask the middleware to disconnect when the bandwidth is fluctuating, or when the battery power is too low. We therefore establish an association between particular context configurations that depend on the value of one or more resources the middleware monitors, and policies that have to be applied;
- active information, where, for every service the middleware delivers, the application specifies the policies that have to be applied and under which environmental circumstances. For example, different context configurations may require the service 'access data' to be delivered differently: a physical copy of data may be preferred when there is a lot of free space on the device, while a link (i.e., network reference) may become necessary when the amount of available memory prevents us from creating a copy, and the network connection is good enough to allow reliable read and write operations across it.

These profiles are then passed down to the middleware. By interacting with the underlying network OS, the middleware maintains an updated representation of the context. Whenever a change in the execution context is detected, the passive part of the profile is consulted to find out which policy must be applied in accordance with the application needs. The active part of the profile is used instead each time the application directly asks the middleware to deliver a specific service. In both situations, middleware *learns* how to behave by looking up at the information the application has passed down to it.

Now the question is whether it is reasonable to assume that the application fixes its own profile once and for all at the time of installation and never changes it after. The answer is no. Both the needs of the user and the context change quite frequently, and we cannot expect the application designers to foresee all the possible configurations. We therefore need to provide the middleware with an initial profile, and then grant the application dynamic access to it. Here is where reflection comes into play.

Reflection. By definition [4], reflection allows a program to access, reason about and alter its own interpretation. The principle of reflection has been mainly adopted in programming languages, in order to allow a program to access its own implementation (see the reflection package of Java or the interface repository in CORBA). The use of reflection in middleware is more coarse-grained and, instead of dealing with methods and attributes, it deals with middleware data and metadata, as stated in [2]. In particular, applications use the reflective mechanisms provided by middleware to create, read and modify their own profile, so that changes in this information immediately reflect into changes in the middleware behaviour. For example, at start-up the application can ask the middleware to access remote data creating a local copy, if there is enough space; later, the application may use reflection to update its profile and instruct the middleware to create a copy under more restrictive conditions, for example, only if there is enough memory and battery left.

4 XMIDDLE: a First Prototype

To prove the suitability of our approach, we have developed XMIDDLE [7], a middleware for mobile computing that focuses on application-driven replication and reconciliation strategies over ad-hoc mobile networks, by means of reflection and metadata.

In particular, XMIDDLE assumes that hosts store their data in a tree structure. On each device, a set of access points for the private tree are defined; they address branches of trees that can be linked (i.e., read and modified) by peers. During disconnections, users continue to update local replicas independently of each other. Upon reconnection, XMIDDLE checks whether the two hosts share a subtree and, if so, a reconciliation process is started. Whenever a conflict is revealed, that is, whenever a difference in the tree is detected, XMIDDLE finds out which reconciliation policy it has to apply on that node by consulting metadata associated to the tree. This meta-information is attached to the tree by the application and can be modified at any time using the reflective mechanism supported by XMIDDLE.

The current implementation of XMIDDLE is based on Java, XML technologies and UDP/IP over WaveLan. In particular, XML has been used to encode metadata, that is, application profiles containing the policies the middleware has to adopt during the reconciliation process. A reflective API allows applications to read and modify this information at run-time, enabling dynamic inspection and adaptation of middleware behaviour.

5 Future Directions of Research

The growing success of new devices and applications for wireless settings calls for the investigations of mobile computing middleware that fit the new scenario. In this paper we have discussed an approach based on the principles of reflection and metadata.

Many issues can be found on our research agenda. First, the communication paradigm we provide at the moment is very rudimental (e.g., sharing of tree), and we may need to extend it in future in order to support more complex interactions. Second, we need to face the problem of inconsistencies: what happens if two hosts ask the middleware to apply different policies during the reconciliation process? Another major issues is security: how can we give access only to authorized users? How can we prevent malicious programs to break into our device and use the reflective mechanism to modify the behaviour of the middleware against us? Once we have found an answer to these basic questions, we plan to extend our initial prototype in order to support the complete reflective model, and test it to evaluate its performance.

References

1. L. Capra, W. Emmerich, and C. Mascolo. Middleware for Mobile Computing: Awareness vs. Transparency (Position Summary). In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, Schloss Elmau, Germany, May 2001. To Appear.
2. L. Capra, W. Emmerich, and C. Mascolo. Reflective Middleware Solutions for Context-Aware Applications. In *Proc. of REFLECTION 2001. The 3^{rd} International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, Japan, September 2001. To Appear.
3. Meta Data Coalition. Open Information Model Version 1.0. http://www.mdcinfo.com/OIM/OIM10.html, 1999.
4. F. Eliassen, A. Andersen, G. S. Blair, F. Costa, G. Coulson, V. Goebel, O. Hansen, T. Kristensen, T. Plagemann, H. O. Rafaelsen, K. B. Saikoski, and W. Yu. Next Generation Middleware: Requirements, Architecture and Prototypes. In *Proceedings of the 7^{th} IEEE Workshop on Future Trends in Distributed Computing Systems*, pages 60–65. IEEE Computer Society Press, December 1999.
5. W. Emmerich. *Engineering Distributed Objects*. John Wiley & Sons, April 2000.
6. W. Emmerich. Software Engineering and Middleware: A Roadmap. In *The Future of Software Engineering - 22^{nd} Int. Conf. on Software Engineering (ICSE2000)*, pages 117–129. ACM Press, May 2000.
7. C. Mascolo, L. Capra, and W. Emmerich. XMIDDLE: A Middleware for Ad-hoc Networking. 2001. Submitted for Publication.