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

Abstract. The increasing popularity of wireless devices, such as mobile phones, personal digital assistants, watches and the like, is enabling new classes of applications that present challenging problems to designers. Applications have to be aware of, and adapt to, variations in the context of execution, such as fluctuating network bandwidth, decreasing battery power, changes in location or device capabilities, and so on. In this paper, we argue that middleware solutions for wired distributed systems cannot be used in a mobile setting, as the principle of transparency that has driven their design runs counter to the new degrees of awareness imposed by mobility. We propose a new middleware model, based on the principle of reflection and metadata as a means for middleware to give applications dynamic access to information about their execution context. Finally, we illustrate a first prototype we have developed which focuses on the problem of data replication and reconciliation in a mobile environment.

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 [10] 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 [9]. 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 [4]. 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

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

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 (Section 2). In Section 3 we describe our mobile middleware based on the principles of reflection [8] and metadata [7], and in Section 4 we briefly illustrate a first prototype we have developed, which focuses on the problem of data replication and reconciliation in a mobile environment. Section 5 discusses and evaluates our work, and Section 6 concludes the paper and lists future work.

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.

3.1 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.

The application profile is written by the application designer and then managed by the underlying middleware, that is, there must be an agreement between the two parts about the representation of the profile. We believe that the eXtended Markup Language (XML)[3], and related technologies (in particular XML Schema [11]) can be successfully used to model this information. In our scenario, middleware defines the *grammar*, that

is, the rules that must be followed to write profiles, in an XML Schema; the application designer then encodes the profile in an XML document that is a valid *instance* of the grammar, according to the definition of valid XML documents given in [11]. Every change done later to the profile must respect the grammar, and this check can be easily performed using available XML parsers [20].

Fig. 1 and 2 illustrates how an application profile looks like when encoded in XML.

```
<RESOURCE name="battery">
    <STATUS operator="lessEqual" value=x/>                % context configuration
    <BEHAVIOUR policy="disconnect"/>                      % policy
</RESOURCE>
```

Fig. 1. XML encoding of a context aware set-up.

Passive Information. Fig. 1 illustrates an XML encoding of a context aware set-up of an application that asks the middleware to disconnect when the battery power is too low. We 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.

```
<SERVICE name="accessData">
    <BEHAVIOUR policy="copy">
        <RESOURCE name="memory">
            <STATUS operator="greaterEqual" value=x/>
        </RESOURCE>
    </BEHAVIOUR>

    <BEHAVIOUR policy="link">
        <RESOURCE name="bandwidth">
            <STATUS operator="greaterEqual" value=y/>
        </RESOURCE>
        <RESOURCE name="memory">
            <STATUS operator="less" value=x/>
        </RESOURCE>
    </BEHAVIOUR>
</SERVICE>
```

Fig. 2. XML encoding of an application service request.

Active Information. Fig. 2 illustrates an XML encoding of an pplication service request, where the service requested is to access remote data that has not been cached locally. This service can be delivered using at least two different policies, 'copy' and 'link'. The application specifies in its profile that a physical copy of data must be preferred when there is a lot of free space on the device, while a link (i.e., network reference) should be performed 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. Therefore, for every service the application may ask the middleware, the application profile specifies the policies that have to be applied and the requirements that must be satisfied in order to choose which of them to apply. These requirements are expressed in terms of the execution context.

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.

3.2 Reflection

By definition [8], 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 [5]. In particular, applications use the reflective mechanisms provided by middleware (what we call a *Reflective API*) to create, read and modify their own profile, so that changes in this information immediately reflect into changes in the middleware behaviour. As we have seen in the previous section, we store application profiles as XML documents. These documents can be semantically associated to trees; the information stored can then be manipulated through the DOM (Document Object Model) API [1], which provides primitives for traversing, adding and deleting nodes to an XML tree, and the XPath [6] technology, which enables the addressing of specific points of an XML document. All the profiles are stored permanently by the middleware, but only the ones of running applications are actually 'active'. When a profile is initially passed down, and whenever an update is invoked, middleware is in charge of verifying the consistency of the information it contains, before storing it permanently. For this purpose, middleware runs a validating parser that parses the document and checks whether it is a valid XML instance of the grammar provided by the middleware to the application. XML Schema supports the specification of very detailed and complex grammars, so that the check performed by the parser can be extremely accurate. In case of an inconsistency, the update fails and an error message is reported to the application. The Reflective API can also be used in a sort of 'expert' mode to update the grammar (the XML Schema definition), that is, the set of services, the set of policies and the set of resources the middleware manages. Middleware is in charge of checking the consistency of the update; for example, if a new policy P is introduced, the code for it must be provided[1].

4 XMIDDLE: a First Prototype

To prove the suitability of our approach, we have developed XMIDDLE [17], 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,

[1] If everything is implemented in Java, the existence of a class P (to be dynamically loaded by the Java Class Loader) can be required.

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 Discussion and Related Work

We have described a middleware for context-aware mobile applications based on the principle of reflection and metadata. Through metadata, we achieve separation of concerns, that is, we distinguish what the middleware does from how the middleware does it. Reflection is then used to provide applications dynamic access to middleware metadata.

The principle of reflection flection has already been investigated by the middleware community during the past years, mainly to achieve flexibility and dynamic configurability of the ORB. Examples include OpenCorba [16], dynamicTAO [15], the work done by Blair et al. [8], etc. Even though we adhere to the idea of using reflection to add flexibility and dynamic configurability to middleware systems, the platforms developed to experiment with reflection were based on standard middleware implementations (i.e., CORBA), and therefore not suited for the mobile environment.

Other middleware systems have been built to support mobility, without using the reflective principle However, we observe that only partial solutions have been developed to date, mainly focused on providing support for location awareness (e.g., Nexus [12] and Teleporting [2]) on one hand, and for disconnected operations and reconciliation of data on the other hand (e.g., Bayou [19] and Odyssey [21]).

Tuple space coordination primitives, initially suggested for Linda [13], have been employed in a number of mobile middleware systems such as JavaSpaces [22], Lime [18], and T Spaces [14], to facilitate component interaction for mobile systems. Although addressing in a natural manner the asynchronous mode of communication characteristic of ad-hoc and nomadic computing, all these systems are bound to very poor data structures (i.e., flat unstructured tuples), which do not allow complex data organization and therefore can hardly be extended to support metadata and reflection capabilities. We believe that XML, and in particular its associated hierarchical tree structure, allows semantically richer data and metadata formatting, overcoming this limitation.

6 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? This problem can be extended to application profiles in general; what happens, for example, if two applications ask the middleware to behave differently when executing in the same context? What if the same application requires conflicting behaviors when changes related to different resources happen at the same time (e.g., “disconnect when battery is low” vs. “connect when bandwidth is high”)? All these questions are currently under investigation.

Another major problem 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. V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. Le Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. Document Object Model (DOM) Level 1 Specification. W3C Recommendation http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, World Wide Web Consortium, October 1998.
2. F. Bennett, T. Richardson, and A. Harter. Teleporting - making applications mobile. In *Proc. of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 82–84, Santa Cruz, California, December 1994. IEEE Computer Society Press.
3. T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language. Recommendation http://www.w3.org/TR/1998/REC-xml-19980210, World Wide Web Consortium, March 1998.
4. 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)*, page 142, Schloss Elmau, Germany, May 2001.
5. L. Capra, W. Emmerich, and C. Mascolo. Reflective Middleware Solutions for Context-Aware Applications. In *Proceedings of REFLECTION 2001. The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, Kyoto, Japan, September 2001. To appear.
6. J. Clark and S. DeRose. XML Path Language (XPath). Technical Report http://www.w3.org/TR/xpath, World Wide Web Consortium, November 1999.
7. Meta Data Coalition. Open Information Model Version 1.0. http://www.mdcinfo.com/OIM/OIM10.html, 1999.
8. 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.
9. W. Emmerich. *Engineering Distributed Objects*. John Wiley & Sons, April 2000.
10. W. Emmerich. Software Engineering and Middleware: A Roadmap. In *The Future of Software Engineering - 22nd Int. Conf. on Software Engineering (ICSE2000)*, pages 117–129. ACM Press, May 2000.
11. David C. Fallside. XML Schema. Technical Report http://www.w3.org/TR/xmlschema-0/, World Wide Web Consortium, April 2000.
12. D. Fritsch, D. Klinec, and S. Volz. NEXUS Positioning and Data Management Concepts for Location Aware Applications. In *Proceedings of the 2nd International Symposium on Telegeoprocessing*, pages 171–184, Nice-Sophia-Antipolis, France, 2000.
13. D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.
14. IBM. T spaces. http://almaden.ibm.com/cs/TSpaces.
15. F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L.C. Magalh aes, and R.H. Cambpell. Monitoring, Security, and Dynamic Configuration with the *dynamicTAO* Reflective ORB. In *International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000)*, pages 121–143, New York, April 2000. ACM/IFIP.
16. T. Ledoux. OpenCorba: a Reflective Open Broker. In *Reflection'99*, volume 1616 of *LNCS*, pages 197–214, Saint-Malo, France, 1999. Springer.
17. C. Mascolo, L. Capra, and W. Emmerich. XMIDDLE: A Middleware for Ad-hoc Networking. 2001. Submitted for Publication.
18. Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. LIME: A Middleware for Physical and Logical Mobility. In *Proceedings of the* 21^{st} *International Conference on Distributed Computing Systems (ICDCS-21)*, May 2001. To appear.

19. K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible Update Propagation for Weakly Consistent Replication. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, pages 288–301. ACM Press, 1997.
20. The Apache XML Project. Xerces Java Parser. http://xml.apache.org/xerces-j/index.html, 2000.
21. M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications*, 3(1):26–33, February 1996.
22. J. Waldo. Javaspaces specification 1.0. Technical report, Sun Microsystems, March 1998.