Towards a Mobile Computing Middleware: a Synergy of Reflection and Mobile Code Techniques

Licia Capra, Cecilia Mascolo, Stefanos Zachariadis and Wolfgang Emmerich
Department of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
{L.Capra|C.Mascolo|S.Zachariadis|W.Emmerich}@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, frequent 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 synergy of reflection and code mobility as a means for middleware to give applications the desired level of flexibility to react to changes happening in the environment, including those that have not necessarily been foreseen by middleware designers. We use the sharing and processing of images as an application scenario to highlight the advantages of our approach.

1. Introduction

In the past decade, middleware technologies 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.

Recent advances in wireless networking technologies, such as WaveLan and Bluetooth, and the growing success of mobile computing devices, such as third generation mobile phones and PDAs, have opened up the door to new classes of distributed applications that impose challenging problems to developers. Wireless devices face temporary loss of network connectivity when they roam; they need to discover other mobile devices in an ad-hoc manner; they are likely to have scarce resources, such as low battery lifetime, slow CPU speed, etc. Applications running on these devices have to be aware of, and adapt to, frequent and unannounced changes happening in their execution environment, such as high variability of network bandwidth, new physical location, and so on.

Middleware solutions developed for wired distributed systems cannot be successfully applied in this scenario, as the principle of transparency that has driven their design runs counter to the new degrees of awareness imposed by mobility. Transparency takes away complexity by allowing middleware to take decisions on behalf of the application. The application, however, can normally make more efficient and better informed trade-off decisions on the use of resources by taking into account application-specific information.

We believe that middleware capable of supporting the development of mobile applications will play a key role in the success of wireless technologies and mobile applications, the same way traditional middleware did for wired distributed systems.

In this paper, we propose a mobile computing middleware that support awareness and high adaptation to context changes, based on the synergistic combination of the principles of reflection [17] and mobile code [9]. We argue that this synergy provides the flexibility that mobile computing middleware must have to cope with both foreseen (through reflection) and unforeseen changes (through code mobility). It improves the ability to exploit scarce resources that mobile devices are often faced with, together with the ability to take advantage of the temporary proximity of services and other hosts.

The paper is organized as follows: in Section 2 we introduce the basic concepts that have driven the design of our middleware, that is, reflection and code mobility. Section 3 describes the conceptual model in details and Section 4 illustrates a mobile computing application in the area of image processing and exchange on which we are working with the support of Kodak. In Section 5 we discuss and evaluate our work and in Section 6 we conclude the paper and list future work.

2. Why Reflection and Mobile Code Techniques

In this section, we introduce the basic principles that have driven the design of our mobile computing middleware.

Applications running on a mobile device need to be aware of, and adapt to, changes in their execution context. By context, we mean everything that can influence the behaviour of an application. Under this general term, we can identify two more specific levels of awareness: *device awareness* and *environment awareness*. Device awareness refers to everything that resides on the physical device that hosts the application; for example, memory, battery power, screen size, processing power and so on. We call these entities *internal resources*. Environment awareness refers to everything that is outside the physical device, that is bandwidth, network connectivity, location, other hosts (or services) in reach, and so on. We call these entities *external resources*.

On one hand, being aware of the execution context requires the designer to know, for instance, the location of the device, the hosts in reach, and, in general, any piece of information that is collected from the network operating system. On the other hand, we do not want the application designers to build their applications directly on the network OS, as this would be extremely tedious and error-prone; therefore, a middleware has to be put in place. The middleware must interact with the underlying network operating system and update 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).

Reflection is the means we provide applications to exploit contextual information and adapt middleware behaviour accordingly. By definition [6], 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. In a middleware context, reflection permits the flow of environmental information, such as the hosts/services currently in reach, the remaining battery lifetime, the location, and the bandwidth conditions, to reach the application layer. Moreover, through reflection the application can instruct the middleware on how to behave in those conditions that the middleware designers consider likely to be unstable.

Code Mobility allows adaptation of middleware behaviour to situations that have not been foreseen at design time. In a mobile ad-hoc setting, middleware designers cannot forecast all the possible context configurations that a mobile application is going to be in. Moreover, application needs may change during its lifetime, and situations that were not considered critical for the system at design time, may later need to be faced using different strategies. It is therefore not feasible to load all possible behaviours on the device, in order to deal with any context configurations, because these behaviours are not known a priori, and anyway this would require a huge amount of memory that portable devices do not have. Mobile code offers a reasonable alternative, which also enhances the use of scarce memory. New behaviors can be delivered from time to time, and downloaded when needed, either from a service provider or from other peers in reach which use the same behaviour, using mobile code techniques. This requires the hosts that are in reach to exchange information about what services, code and resources are available, in order to download protocols and exploit resources. Reflection can be combined with mobile code techniques to allow applications to select, for example, from where to download protocols, based on application-specific information (e.g., choose the most trusted host, the nearest site, etc.).

To illustrate how reflection and code mobility can be combined, we can consider a simple but effective example. Middleware designers may offer applications two different behaviours to access remote data: a ‘copy’ behaviour that replicates remote data locally and work on local copies; and a ‘link’ behaviour that creates a network reference to the master copy and then accesses it remotely. Applications use reflection to dynamically adapt middleware behaviour to their needs; for example, they can instruct middleware to copy data in conditions of high memory availability and low bandwidth, and to link data when network bandwidth is instead high and stable, but the host is running out of memory. The application may then find itself in a situation of high memory and high, but expensive, network bandwidth (e.g., GSM charges users for the amount of time they remain connected, and UMTS for the amount of network traffic produced). The copy procedure may not be suitable anymore if we have to transfer a considerable amount of data, as it requires time, ad therefore money. A smarter behaviour would be to transfer compressed data, in order to reduce both the downloading time and the traffic moved around. However, if this situation has not been foreseen, middleware does not have locally this behaviour, and it might be necessary to obtain the newly delivered protocol implementation from peers in reach.

Thus, the combination of reflection and mobile code enables middleware to cope with both foreseen and unforeseen changes in the context. In the next section, we illustrate how to use these principles in practice.

3. Our Mobile Computing Middleware

In this section, we illustrate how reflection and mobile code techniques are integrated in our mobile computing middleware.

In order to allow applications to dynamically change middleware behaviour, and to let the middleware understand and apply this behaviour, the two layers have to agree on a specific format for this information. We believe that the eXtended Markup Language (XML)[5], and related technologies (in particular XML Schema [7]) can be successfully used to encode what we call *application profile*, that is, meta-information that, for each application, associates policies (i.e., middleware behaviour) to context configurations. In our scenario, middleware dictates the *grammar*, that is the rules that must be followed to write profiles, in an XML Schema definition; the application designer then encodes the profile in an XML document that is a valid *instance* of the grammar. Every change done later to the profile must respect the grammar, and this check can be easily performed using available XML parsers.

To understand what information to encode, we distinguish two different ways in which the application influences the behaviour of the middleware.

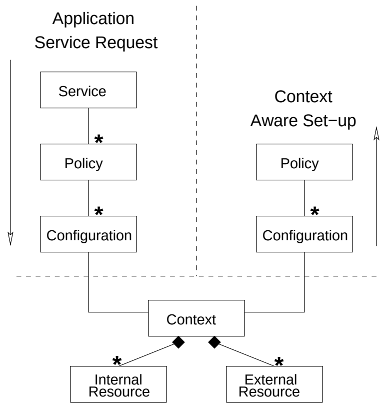

Figure 1. Application profile.

1. *Changes in the execution context.* The application can ask the middleware to listen to changes in the execution context and 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 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, as shown on the right-hand side of Figure 1. Figure 2 illustrates a simple example of an XML document for this kind of information.

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

Figure 2. XML encoding of a context aware set-up.

Middleware interacts with the underlying network operating system in order to keep an updated configuration of the context. Whenever a change in the context happens, it looks up in the application profiles of running applications whether one or more of them have registered an interest in the changed resources, and triggers the corresponding actions.

2. *Service request.* The application can ask the middleware to execute a service; for example, to access some remote data it has not cached locally. There are many different ways a service can be provided; for example, the service ‘access data’ can be delivered using at least two different policies, ‘copy’ and ‘link’, as discussed in the previous section. The circumstances under which an application may want to use them are different: a physical copy of data may be preferred when there is a lot of free space on the device, while a link 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. 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 (left-hand side of Figure 1). Figure 3 gives an example of how to express this information in the application profile using XML.

For the reflective principle, middleware must grant applications dynamic access to their profiles. A reflective API is therefore provided that exploits the Document Object Model (DOM) [1] API to give applications access to the tree representation of their profile. Whenever a profile is modified, the 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 (e.g., it checks whether the selected policy is contemplated in the Schema definition).

```
<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>
```

Figure 3. XML encoding of an application service request.

As we have argued in Section 2, reflection allows applications to react to foreseen changes in the environment; however, particularly in ad-hoc settings, there may be many unforeseen changes that require the adoption of new behaviours that were not considered at design time. We therefore extend our model with the use of mobile code techniques. Code mobility provides higher degrees of flexibility as it allows hosts to dynamically download new behaviours when needed. It is now necessary to specify, for each policy, where the middleware can find the code of the corresponding algorithm, in case it is not already on the device and therefore it must be downloaded on demand from one of the hosts in reach, if possible [1].

Figure 4 shows how the application profile defined in Figure 3 can be refined to support the new model.

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

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

Figure 4. XML encoding of an application service request with code mobility specification.

Note that the model is very general as it allows the application to specify, for instance, from which 'kind' of hosts the behaviour can be fetched. For example, in Figure 4 the profile explicitly states that the 'copy' protocol can be fetched from the nearest host that has it. As for the linking behaviour, the '*' key is used to specify that the middleware is free to load the code from any host in reach. Other possibilities include the definition of a specific IP address (to select, for example, a trusted service provider) or keywords that identify classes of hosts, such as 'PC' [2].

The discovery of new policies happens during peer-to-peer interactions. First, base stations advertise to the hosts in reach new available policies for specific services; this means that the XML Schema definition on the hosts in reach is updated to include the name of the new behaviours. However, we do not require the hosts to load the corresponding code too, as it may not be possible immediately due to memory limitations. Mobile hosts may then move away and form ad-hoc networks with other hosts; during interactions, the Schema definition of these peers is updated. However, as the code is not carried around all the time, it is possible for the application to write valid profiles which specifies policies not available locally. If that happens, the middleware can exploits the meta-information contained in the profile to fetch it.

In the next section, we show an industrial application scenario in the area of image sharing, collaboration and uploading, which we use to describe the use of our approach in a mobile setting.

4. An Application Scenario

In this section we present an example in the context of image processing and image based services that we are partly developing with Kodak. Our goal is to illustrate how reflection and mobile code techniques may provide the flexibility required by mobile computing middleware.

Let us consider a number of mobile devices (cameras, smart phones, PDAs and laptops) that are able to communicate through an ad-hoc network infrastructure. Some base stations equipped with fixed network connectivity, Internet facilities and high resolution displays are also present. Mobile devices store pictures in their memory. Pictures may be shared among devices and may be displayed and shipped through the Internet to base stations, in order to be permanently stored in some available database (Figure 5).

The sharing, displaying, shipping and uploading of the images may happen in many potentially different ways, according to the current network availability and the devices involved, as they greatly vary in terms of memory, processing power, etc. Different compression, rendering, loading, shipping, linking, caching protocols may have to be used by the middleware.

[1] Some default behaviours are defined in order to handle "code not found" exceptions.

[2] It is up to the middleware layer to have some ways of resolving the name 'PC'.

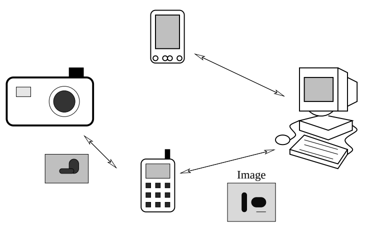

Figure 5. The image sharing scenario.

In this scenario, the middleware cannot transparently choose which protocol to apply based on the current configuration of the execution context, as the applications possess vital knowledge that must be exploited during this choice. For example, an application may want to persistently store on a remote database only a specific subset of the images the device is carrying, in order to save battery power, and the middleware has no way to guess this subset. We believe reflection can be used to allow applications to inspect contextual information and to change middleware behaviour accordingly at run-time. In this way, applications have a say in which way images are handled, shipped, cached or linked.

However, reflection alone is not enough. The devices are not able to know a-priori which protocols they are going to need in which situation, and which other devices they are going to encounter. Devices cannot load all the possible behaviours into memory, due to memory constraints; moreover, new behaviours may be delivered from time to time to cope with unforeseen context configurations and new application needs. Therefore, reflection and mobile code techniques have to be combined.

Figure 6 illustrates how an application can influence middleware behaviour when loading and displaying an image. As discussed in Section 3, the profile contains two essential types of information: which policy the middleware has to apply when executing in a particular context configuration, and from where to fetch the code of the policy, if not found locally.

As the picture shows, the first policy describes how the loading of an image should happen when memory is relatively high. The protocol that must be applied (i.e., 'load', as specified by the tag `CODE`) can either be found on the host or fetched remotely from *any* peer (as shown by the use of the '*' key). The second policy defines a way of loading and seeing the picture in fragments, when memory is low. In this case, if the code for the `fragmentedload` cannot be found locally, it can only be fetched from the host corresponding to the specified `IPaddress`, for example, a trusted base station. The third policy allows the load and display of an image on a more powerful host, if there is one in reach and the network bandwidth is good enough to allow the shipping. Protocols for proper decom-

```
<SERVICE name="displayPicture">
    <BEHAVIOUR policy="load">
        <RESOURCE name="memory">
            <STATUS operator="greaterEqual" value=x/>
        </RESOURCE>
        <CODE name="load" fromhost="*"/>
    </BEHAVIOUR>

    <BEHAVIOUR policy="fragmentedload">
        <RESOURCE name="memory">
           <STATUS operator="less" value=x/>
        </RESOURCE>
        <CODE name="fragmentedload" fromhost=IPaddress/>
    </BEHAVIOUR>

    <BEHAVIOUR policy="remote">
        <RESOURCE name="bandwidth">
            <STATUS operator="greaterEqual" value=y/>
        </RESOURCE>
        <RESOURCE name="remotedisplay">
            <STATUS operator="inreach" />
        </RESOURCE>
        <CODE name="remote" fromhost="nearestHost"/>
        <UPLOAD>
            <CLASS name="decompress"/>
            <CLASS name="decrypt"/>
        </UPLOAD>
    </BEHAVIOUR>
</SERVICE>
```

Figure 6. XML encoding of policy settings in the image processing example.

pression/decryption of the image may need to be uploaded with the image to the remote host as well. These protocols can be specified in the behavioral description under the tag `UPLOAD`.

More complex situations can be modeled, where the behaviour of the middleware depends not only on the current context configuration but also on the particular data it is managing. Pictures, in fact, have a header of meta-data describing picture details and compression characteristics. Applications could exploit this information to write more sophisticated profiles in which they specify which compression/cropping/encryption modules the middleware should use to handle a particular type of image, instead of moving high resolution images around with no need.

5. Discussion and Related Work

We have described a middleware for mobile computing based on the principle of reflection and code mobility. Through reflection, we allow applications to adapt middleware behaviour dynamically, but only in the case of foreseen situations. Mobile code techniques add new level of flexibility as they allow adaptation of middleware to unforeseen situations too; moreover, they support a better exploitation of scarce resources (e.g., memory) typical of hand-held devices.

The use of reflection and mobile code in the context of mobile computing is however not new. The principle of reflection 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 [13] the work done by Blair et al. [6], 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.

Researchers have also focused on how to use code mobility to exploit mobile devices capabilities and constrained conditions better. Some work in this direction has been carried out in [3, 15, 14], where mobile agents [12, 27], a specialized version of mobile code with autonomous components, are used in mobile computing settings to overcome the problem of intermittent network connectivity.

We believe that not only agents but mobile code in general can be useful in mobile computing and that code mobility and mobile computing research areas are more and more destined to meet.

Other middleware systems have been built to support mobility, without using the reflective principle or code mobility. However, we observe that only partial solutions have been developed to date, mainly focused on providing support for location awareness (e.g., Nexus [8] and Teleporting [4]) on one hand, and for disconnected operations and reconciliation of data on the other hand (e.g., Bayou [21] and Odyssey [22]).

Tuple space coordination primitives, initially suggested for Linda [10], have been employed in a number of mobile middleware systems such as JavaSpaces [26], Lime [20], and T Spaces [11], 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. Conclusions and Future Directions

The growing success of mobile computing devices and networking technologies, such as WaveLan [25] and Bluetooth [19], call for the investigation of new middleware that deal with mobile computing requirements, in particular with context-awareness. Our goal in this paper has been to outline a global model for the design of mobile middleware systems, based on the principle of reflection and mobile code techniques.

The choice to use XML to represent metadata comes from our previous experience with XMIDDLE [18], an XML-based middleware for mobile systems that focuses on data reconciliation and synchronization problems and solves them exploiting application-specific reconciliation strategies. Our plan is to extend the previously built prototype to fully support the model presented here. At present, the middleware interaction with the network operating system in order to capture contextual information is quite limited in our prototype; however we have clear ideas on how this could be extended using for example, the Mobile Information Device Profile (MIDP) [24], that is part of the Java 2 Platform Micro Edition [23]. MIDP is a set of Java APIs which provide a runtime environment targeted to mobile information devices, such as cellular phones and PDAs. The MIDP specification addresses issues such as user interface, persistence storage, networking, and application model.

We have discarded the idea of implementing our model upon Jini [2], an architecture that allows groups of devices and software components to federate into a single, dynamic distributed system, enabling dynamic discovery of services inside the network federation. The reason for this choice is that Jini targets nomadic networks; it assumes in fact the existence of a core of fixed hosts which provides mechanisms for devices, services and users to join and detach from a network in an easy and natural, often automatic, manner. Moreover, it relies on the existence of a network of reasonable speed connecting Jini technology-enabled devices that have some memory and processing power. Although we have given in this paper an example of application for a nomadic setting, we do not want to constraint ourselves to this scenario. We therefore think that XMIDDLE represents a better starting point for our needs, as it targets pure ad-hoc networks.

Apart from extending our prototype, our plans for the future include more fundamental research issues. In this paper the main form of code mobility we have discussed is “code on demand”; however, we believe that the combination of different mobile code techniques could enhance our model. In some cases, for example, migrating the computation running on a mobile device to a close and more powerful host could improve performance. We are also working on some infrastructure for advertisement of “resource leasing”, so that application code can move among hosts in reach to exploit more powerful resources in an ad-hoc manner. XMIDDLE already supports the exchange of information, such as services, code and resources available, between peers, so that hosts in reach are able to exchange protocols and exploit resources.

Another major issue is safety. What happens 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”)? What if the behaviour requested cannot be found neither locally nor on the hosts in reach? All this questions are currently under investigation.

In a context of mobile code, security plays a very important role. In the model and prototype we developed until now security issues are not addressed. We however plan to look into them

Acknowledgements We would like to thank Kodak and in particular Karen Lawson for providing the application scenario illustrated in the paper. We also thank Jon Crowcroft and Richard Gold for the helpful discussion on some of the ideas described in this paper.

References

[1] V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. L. 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, Oct. 1998.

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

[3] P. Bellavista, A. Corradi, and C. Stefanelli. Mobile Agents Middleware for Mobile Computing. *IEEE Computer*, 34(3):73–81, 2001.

[4] 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, Dec. 1994. IEEE Computer Society Press.

[5] 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, Mar. 1998.

[6] 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, Dec. 1999.

[7] D. C. Fallside. XML Schema. Technical Report http://www.w3.org/TR/xmlschema-0/, World Wide Web Consortium, Apr. 2000.

[8] 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.

[9] A. Fuggetta, G. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5), 1998.

[10] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, 1985.

[11] IBM. T spaces. http://almaden.ibm.com/cs/TSpaces.

[12] J. Kiniry and D. Zimmerman. A Hands-On Look at Java Mobile Agents. *IEEE Internet Computing*, 1(4), 1997.

[13] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. M. aes, and R. 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, Apr. 2000. ACM/IFIP.

[14] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, and G. Cybenko. Agent Tcl: Targeting the Needs of Mobile Computers. *IEEE Internet Computing*, 1(4):58–67, 1997.

[15] E. Kovacs, K. Rohrle, and M. Reich. Integrating Mobile Agents into the Mobile Middleware. In *Proc of the 2nd Int. Workshop on Mobile Agents (MA’98)*, volume 1477 of *LNCS*, pages 124–135. Springer, 1998.

[16] T. Ledoux. OpenCorba: a Reflective Open Broker. In *Reflection’99*, volume 1616 of *LNCS*, pages 197–214, Saint-Malo, France, 1999. Springer.

[17] P. Maes. Concepts and Experiments in Computational Reflection. In *Proceedings of OOPSLA ’87*, pages 147–155, Orlando, Florida, Oct. 1987. ACM Sigplan Notices.

[18] C. Mascolo, L. Capra, and W. Emmerich. An XML-based Middleware for Peer-to-Peer Computing. In *Proc. of the International Conference on Peer-to-Peer Computing (P2P2001)*, Linkopings, Sweden, Aug. 2001. To appear.

[19] R. Mettala. Bluetooth Protocol Architecture. http://www.bluetooth.com/developer/whitepaper/, Aug. 1999.

[20] A. L. Murphy, G. P. Picco, and G.-C. 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.

[21] 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.

[22] M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications*, 3(1):26–33, Feb. 1996.

[23] I. Sun Microsystem. Java 2 Platform, Micro Edition. http://java.sun.com/j2me/, 2000.

[24] I. Sun Microsystem. Mobile Information Device Profile (MIDP) Specification. http://java.sun.com/products/midp/, 2000.

[25] L. Technologies. WaveLan. http://www.wavelan.com, 2000.

[26] J. Waldo. Javaspaces specification 1.0. Technical report, Sun Microsystems, March 1998.

[27] D. Wong, N. Paciorek, and D. Moore. Java-based Mobile Agents. *Communications of the ACM*, 42(3):92–102, 1999.