Middleware for Mobile Computing: Awareness vs. Transparency

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

Abstract

In this paper we argue that middleware solutions for wired distributed systems cannot be used in a mobile setting. We show that mobile applications impose new requirements that run counter to the principles on which current middleware systems have been built. We propose the use of reflection capabilities and meta-data to pave the way for a new generation of middleware platforms designed to support mobility.

1 Introduction

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 application designers. These devices face temporary 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 small amounts of memory; and they are required to react to frequent and unannounced changes in the environment, such as high variability of network bandwidth.

When developing distributed applications, designers do not have to deal explicitly with problems related to distribution, such as heterogeneity, scalability, resource sharing, and the like. *Middleware* developed upon network operating systems provides application designers with a higher level of abstraction so that they can concentrate on what they are really interested in.

In order to address the new complexity arising from mobility, suitable middleware must be designed to offer appropriate support for developing mobile applications. Most of the solutions to date are targeted to satisfy the needs of particular applications and are insufficiently generalized to be reusable.

The main aims of this paper are: to highlight, through a case study, the requirements imposed by mobile computing applications; to argue that middleware solutions for wired distributed systems are not well suited to the mobile setting, as they do not allow dynamic adaptation of the middleware behaviour to changes in the context of execution, and that hence new middleware capabilities are required. Finally, we propose the marriage of reflection and meta-data as the starting point for developing a new generation of middleware systems for mobile computing.

2 Middleware for Wired Distributed Systems

Middleware technologies have been adopted in order to provide the application designer with a higher-level of abstraction, hiding away the complexity introduced through distribution. Existing middleware technologies, such as transaction-oriented, message-oriented or object-oriented middleware [5] have been built adhering to the metaphor of the *black box*, i.e. the existence of many distributed components is hidden from both users and software engineers, so that the system appears as a single integrated computing facility. In other words, distribution is rendered *transparent* [1].

These technologies have been designed and are successfully used for stationary distributed systems built with wired networks, but are they suitable for a mobile setting? The answer seems to be no. 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. Moreover, object-oriented middleware systems, such as CORBA [11], mainly support synchronous point-to-point communication with at-most-once semantics, while in a mobile environment it is frequently the case that client and server hosts are not connected at the same time,

causing this form of communication to become almost useless. Secondly, and most notably, completely hiding the implementation details from the application becomes both more difficult and makes little sense. Mobile systems need to detect and adapt to drastic changes happening in the environment, such as changes in connectivity, bandwidth, battery power and the like.

In order to address the new needs of mobile computing applications, we cannot simply apply existing middleware technologies to this setting, to do so would risk poor performance, high operating costs, unusability and non-scalable solutions. It now seems necessary to demolish one of the foundations on which previous middleware solutions have been built, that is transparency, in favour of a new form of *awareness* that allows application designers to *inspect* the context and *adapt* the behaviour of middleware accordingly.

3 Mobile Computing Applications: a Case Study

In this section we present a collaborative electronic shopping system that we discussed, and partly developed, with Unipower, an industrial collaborator of our research group. Our goal is to highlight the different degrees of awareness needed in a mobile setting. We argue that, 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. This is particularly true in mobile computing settings, where the 'context' (for instance the location) of a device should be taken into account.

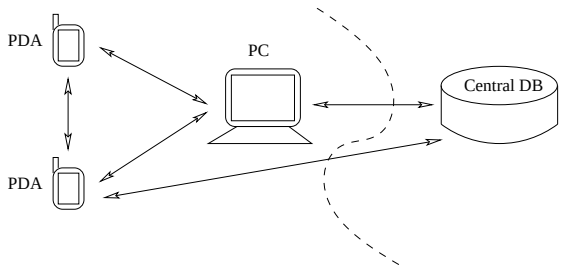

Figure 1: Parties involved in the e-shopping system.

Figure 1 illustrates our case study. Two main parties are involved: the shop, represented by the central database, and a set of customers, represented by PCs and mobile devices. We assume that the shop stores its product catalogue in a centrally administered database and makes it accessible to its customers for consultation at any time. As a possible set of customers we consider a couple that each own a PDA and share a home PC. This couple does its weekly shopping electronically: they use the PC when they are at home and their PDAs while roaming around town.

There are two main operations that our application must support: consulting the product catalogue and determining an order. We now discuss the details of these operations in order to show the new requirements imposed by mobility.

Consulting the product catalogue. A PC and a PDA have different characteristics, especially in terms of battery lifetime, processing power, local memory and quality of the network connection. While it is feasible (at least in principle) for the client application running on the PC to connect to the network each time an access to a category or to a product in the catalogue is needed, this is not affordable in case of a PDA or a mobile phone, due to the high cost and low quality of the network connection. Even for a PC, going through the wire frequently decreases the quality of the service significantly, as a result of network latency.

In order to deal with this problem, the most frequently adopted solution is to replicate the product catalogue locally and update it whenever a price or the portfolio of the shop changes. Even in this situation, PCs and PDAs exhibit rather different characteristics: while a PC generally has a huge amount of disk space that makes it a reasonable strategy to replicate the full product catalogue, the PDA has a limited amount of resources so that it is possible to replicate only *part of* the catalogue. Each time the client application tries to access a product that has not been replicated locally, a request must be sent through the network to the PC or directly to the central database, in order to gain access to it. To avoid performance degradation, we do not want to use these low quality links, which can be as slow as 9,600 bauds for GSM, frequently. Moreover, we would like to be able to continue working even when we have no network connection, that is, the client application does not want to

suffer from network disconnections. It is therefore necessary to 'prefetch' the data the client wants to access and to do so we need to answer the question about *which* information we should replicate.

The replication policies adopted so far by many middleware systems include, among the others, LRU (Last Recently Used) data and MFU (Most Frequently Used) data: in all cases, it is the middleware that decides which information to replicate. If replication is transparent, the client application has no way to influence this choice, even though only it knows what it wants to access. We claim that, instead of letting the middleware guess what the client is going to need, the application has to be able to instruct the middleware on what to 'cache', that is, we call for *replication awareness* instead of *replication transparency*.

Determining an order. Apart from the product catalogue, the application provides the abstraction of a shopping basket that is kept on the PC and is replicated on each PDA. The couple can do its weekly shopping independently of each other and synchronize the shopping basket only from time to time, or directly at the end of the week before submitting an order to the central database. As we want to submit a unique joint order, we need to reconcile the two different (possibly conflicting) versions. This is not a trivial problem. Having two different shopping baskets does not necessary mean having a conflict to resolve (i.e., the parties may order products in different categories). Even when a conflict exists, there is no obvious way to remove it and return to a consistent state. Only the application has the knowledge necessary to point out conflicts and determine ways to fix them. It is therefore the application (and not the middleware) that has to drive the reconciliation process.

Once the shopping basket has been successfully reconciled, we may want to submit the order to the shop. This can be done from the PC at home, using a reliable and high bandwidth connection, or from a PDA while moving around, experiencing high variability in the quality of the network connection. The submission of the order is a critical operation and we want to make sure that the transaction will be successfully completed. There are a number of protocols we may want to use for this purpose, for instance two-phase commit or three-phase commit; while the second is the safest, it is also the heaviest. The middleware cannot opt for one protocol instead of the other by simply testing the condition of the network. This decision strictly depends on the non-functional requirements of the application; if reliability is our primary goal, three-phase commit is a forced choice. If we also care about performance and we do not want to overload the network as well as the server, we may want to use the lighter protocol and, if experiencing too high bandwidth variability, even delay the order until better network connectivity is detected. Information about the environment is also useful to the application to decide whether to work on-line or off-line. In all these situations, middleware cannot take decisions independently from the application in a transparent way. A new form of *context-awareness* is therefore needed.

Finally, while moving it may happen that we want to submit an order and collect the products purchased within short time directly from the nearest branch of the shop. This could be done in a transparent way, with all the orders from all different customers being sent to the central database, and from there redirected to the customer's nearest branch. This solution however causes many problems: for example, the central database can become a bottleneck, causing heavy delays. Secondly, we could not be able to connect to the central database because of a network failure, while still being able to reach one of its branches. A better solution would be to have the application know exactly where it is, locate the nearest shop, and then send the order there. In other words, the application needs to know its own physical location as well as the physical location of the component (the branch) providing the service it is looking for. Another foundation of traditional middleware systems, that is *location transparency*, may have to be replaced with *location awareness*.

We can now abstract away from this particular example and summarize the many different levels of awareness needed by mobile applications: replication awareness against replication transparency, in order to decide what to store locally and what to access remotely; context awareness against context transparency, to react properly to frequent changes of the execution context; and location awareness instead of location transparency. Middleware solutions for mobile applications should be able to support all of these requirements, in order to promote reusable and principled solutions, as opposed to ad-hoc ones.

4 Existing Mobile Computing Middleware

The research area on middleware systems for mobile computing applications is new and there are no well-established results [14]. The topic attracts the interest of many researchers as shown by the increasing number of projects in this field. However, we observe that only partial solutions have been developed to date, largely by modifying existing middleware or by building ad-hoc, application-specific ones.

As we discussed in Section 2, middleware for wired distributed systems do not appear to be suited in mobile settings. In order to cope with this limitation, one strand of middleware research has extended existing middleware for mobile computing. OpenCorba [9], for example, is a reflective open broker that enables users to adapt

dynamically the representation and the execution policies of the software bus. This is achieved through two mechanisms: first, separation of concerns, to distinguish between what an object does (base level) and how the object does it (meta level); this increases readability, re-usability and code quality. Second, objects can change their classes at run-time; we can then dynamically add and remove class properties without having to regenerate the code. This makes it possible to introduce new semantics to the initial model, such as concurrency, replication, security, and so on. Unfortunately, this solution is not targeted to a mobile setting, as the basic mechanisms are the same as in CORBA, and therefore they suffer from the limitations we discussed in Section 2.

In Bayou [12], database applications in mobile computing settings are targeted. The system handles disconnection and reconciliation in a very transparent way, exposing as little as possible to the application that has no way to influence the outcome of the reconciliation process. Although this vision might adapt to some scenarios, a higher level of context awareness is needed to target a broader range of applications.

In Odissey [15], a distributed file system is shared among mobile devices. Files are copied and their content is reconciled when the mobile hosts get in contact with each others using some form of application-specific adaptation. The abstraction of a file as the mobile data unit seems to be too coarse-grained in a context where bandwidth is so scarce and network connection so expensive. Furthermore, a file is nothing but a bytestream, a semantically poor structure that can only partially and with great difficulty exploited to obtain application-specific reconciliation.

More recent systems attempt to cover issues such as cooperation and service availability. Again, being aware of what services and what hosts are around is an important notion in mobile environments. In Jini [2] a global service look-up system is implemented. However, this requires knowledge of configuration of the current context which may be difficult to acquire and maintain.

Tuple space coordination primitives, that were initially suggested for Linda [7], have been employed in a number of mobile middleware systems such as Jini/JavaSpaces [17], Lime [13], and T Spaces [8], to facilitate component interaction for mobile systems. Tuple space based systems exploit decoupling in time and space of the data structures in a context where connections and disconnections are very frequent operations. 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 do not adapt to all the potential application domains. Furthermore, as tuple spaces are multi-sets, systems based on tuple space paradigms also suffer from synchronization limitations; two tuples with same name can co-exist in the same tuple-space (due to multi-set properties), thus if two mobile devices wish to synchronize the middleware needs to force an 'unnatural' operation to 'merge' the tuples.

Blair et al. [4] has defined a "manifesto" for the next generation of middleware systems, claiming that they should be run-time configurable and allow inspection and adaptation of the underlying software. The approach they propose is based on components to structure the middleware platform, and on reflection to inspect and adapt the behavior of these components. The middleware appears to the application designer as a set of customizable components which can be tailored to the needs of the application. Unfortunately, the middleware platform they developed to experiment with reflection was based on a CORBA implementation, therefore not suited for the mobile environment, as already stated.

5 Reflection and Meta-Data: a Possible Solution

In this section we set out the ideas we plan to investigate with a view to providing a generalized middleware solution for mobile computing applications. In order to reach this goal, we need to trade-off between: on one hand, a solution that eases the burden on application designers, leaving most of the control to the middleware; on the other hand, allowing the application to access information about the execution context and influence the way the underlying middleware behaves. We do not want for example the application to have to test the status of the battery power or of the network connection, the middleware has to take care of this, but it has to provide the application with a way to obtain this information whenever needed.

In our approach, *reflection* is the means by which middleware can give dynamic access to information about the application's execution context to the application itself. *Meta-data* is the paradigm we use to implement this. We distinguish between two different levels: a *base-level* of application-specific information managed by the application itself, and a *meta-level* managed by the middleware to structure information usually hidden, that is, network connectivity, available memory, available resources, and so on. Reflection primitives must be provided from the underlying middleware to the application layer in order to allow inspection of this meta-data, and adaptation of the middleware behaviour through manipulation of the meta-data itself.

We have started developing XMIDDLE [10], a middleware for mobile computing in which we are investigating the use of meta-data (represented in the eXtended Markup Language, XML [3]), to structure the information provided

by the middleware to the application. XMIDDLE focuses on synchronization of replicated XML documents. In order to enable application-driven conflict detection and resolution, XMIDDLE supports the specification of conflict resolution policies through meta-data definition using the XML meta-level (called XML Schema [6]). In [16] some formal work on application independent reconciliation has been carried on, which also focuses on a structured way in which applications can influence data reconciliation choices.

The first results we have obtained experimenting with XMIDDLE are promising: XML seems to be a powerful means to structure and manipulate meta-data. Our plan is to extend XMIDDLE in order to fully support these new ideas. To achieve this, we first need to find a structured way to define the set of elements of the execution context that we wish the mobile application to be aware of. Then, we need to find a consistent way to model these aspects with meta-data as well as to allow access to them.

Acknowledgements. We would like to thank our industrial partner Unipower and Ewan Harrow for providing the case study discussed in the paper, Kumaresan Sanmugalingam for the helpful discussions on the topic and Anthony Finkelstein for pointing out the importance of meta-data.

References

[1] ANSA. The Advanced Network Systems Architecture (ANSA). Reference manual, Architecture Project Management, Castle Hill, Cambridge, UK, 1989.

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

[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] 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. 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.

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

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

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

[9] T. Ledoux. OpenCorba: a Reflective Open Broker. In *Reflection'99*, volume 1616 of *LNCS*, Saint-Malo, France, 1999. Springer.

[10] C. Mascolo, W. Emmerich, and L. Capra. XMIDDLE: An XML based Middleware for Mobile Computing . Technical Report RN/00/54, University College London, Dept. of Computer Science, 2000.

[11] T. Mowbray and R. Zahavi. *The Essential CORBA: Systems Integration Using Distributed Objects*. Wiley, 1995.

[12] 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.

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

[14] G.-C. Roman, A. L. Murphy, and G. P. Picco. A Software Engineering Perspective on Mobility. In A. C. W. Finkelstein, editor, *Future of Software Engineering*. ACM Press, 2000.

[15] M. Satyanarayanan. Mobile Information Access. *IEEE Personal Communications*, 3(1), February 1996.

[16] M. Shapiro, A. Rowstron, and A. Kermarrec. Application-independent Reconciliation for Nomadic Applications. In *Proceedings of European Workshop:"Beyond the PC: New Challenges for the Operationg System"*, Kolding, Denmark, 2000. SIGOPS.

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