Exploiting Reflection in Mobile Computing Middleware

Licia Capra[a] **Gordon S. Blair**[b] **Cecilia Mascolo**[a]
Wolfgang Emmerich[a] **Paul Grace**[b]
[a]Department of Computer Science, University College London, London, UK
[b]Distributed Multimedia Research Group, Computing Department, Lancaster University, Lancaster, UK

The increasing popularity of portable devices and recent advances in wireless networking technologies facilitate the engineering of new classes of applications, which present challenging problems to designers. Mobile devices face temporary and unannounced loss of network connectivity when they are moved, they are likely to have scarce resources, and they are required to react to frequent changes in the environment. To accommodate these new requirements imposed by mobility, middleware platforms for mobile computing must be capable of both deployment-time configurability and run-time reconfigurability. We illustrate how reflective techniques can be exploited by middleware designers to address these requirements. We discuss two complementary approaches: CARISMA, where reflection is used to support dynamic adaptation of middleware behaviour to changes in context, and ReMMoC, which uses reflection to accommodate heterogeneity requirements imposed by both applications and underlying device platforms.

I. Introduction

The increasing popularity of portable computing devices, such as mobile phones, personal digital assistants, digital cameras and the like, and recent advances in wireless networking technologies are facilitating the construction of new classes of distributed and mobile applications. These applications pose a number of new challenges for middleware technology. In particular, the middleware that supports the construction of such applications has to be *light-weight*, in order to run on resource-constrained devices; middleware has also to *support context-awareness*, as mobile applications need to 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. Finally, the problem of *interoperating with heterogeneous middleware* technologies that exist in different locations needs to be overcome; for example, middleware might provide different asynchronous communication paradigms that may be used to cope with the frequent and unannounced disconnections that are typical of the mobile environment.

Unfortunately, the current generation of mainstream middleware is, to a large extent, heavyweight, monolithic and inflexible and, thus, fails to properly address these requirements. The main reason is that traditional middleware systems have been built adhering to the principle of transparency: implementation details are hidden from both users and application designers and are encapsulated inside the middleware itself, so that the distributed system appears to application developers as a single integrated computing facility. Although having proved successful for building traditional distributed systems, this approach suffers from severe limitations when applied to the mobile setting where it is neither always possible, nor desirable, to hide all the implementation details from the user. Such black box systems are inevitably heavyweight, have built-in mechanisms and policies that are not suitable for mobile computing and cater for the common case rather than the high levels of heterogeneity intrinsic in mobile environments. In addition, applications may have valuable information that could enable the middleware to execute more efficiently.

In order to cope with these limitations, many research efforts have focused on designing new middleware systems capable of supporting the requirements imposed by mobility (see section II). As a result of these efforts, a pool of mobile middleware systems has been produced. However, none of the solutions developed to date supports the necessary level of middleware configurability and reconfigurability that is required to accommodate mobile computing. In our opinion, a more systematic, principled and dynamic solution is needed.

We hypothesise that *reflection* [41] offers significant advantages for building mobile computing middleware. Reflection is a principled technique support-

ing both introspection and adaptation (see section III). The technique hence supports more configurable and reconfigurable middleware (e.g., [5]). This means that a middleware core with only a minimal set of functionalities can be installed on a mobile device. Context information can be maintained by the middleware and through reflective mechanisms applications can acquire information about their execution context and tune the middleware behaviour accordingly. No specific communication paradigm is related to the principle of reflection, so this issue is left unspecified and depends on the particular middleware system built.

The remainder of the paper is structured as follows: Section II describes the state of the art of mobile computing middleware and points out gaps in current approaches. Section III defines the concept of reflection, and how it applies to mobile computing middleware. In Section IV, two projects that exploit reflection to build mobile computing middleware are presented: CARISMA (Context-Aware Reflective mIddleware System for Mobile Applications), that aims at enhancing the development of context-aware applications, and ReMMoC (Reflective Middleware for Mobile Computing), that aims at overcoming the problem of heterogeneity in mobile middleware technology. We then briefly discuss related work before concluding the paper and pointing out some future open issues.

II. Mobile Computing Middleware

In this section we look in turn at state of the art middleware, including traditional middleware types used in the mobile domain and new classes of middleware built specifically for mobility. However, an exhaustive review of all existing mobile computing middleware is beyond the scope of this paper and we refer the interested reader to [27].

II.A. Existing Middleware Standards and Platforms

There have been attempts to adapt traditional object-oriented middleware, including CORBA, DCOM and Java RMI, to mobile settings, mainly to allow mobile devices to interoperate with services within existing fixed networks. For example, IIOP (the Internet Inter-ORB Protocol) has been successfully ported to the mobile setting, presenting a layered architecture that ensures an IIOP connection remains established transparently [19]. Similarly, RAPP [39] and DOLMEN [25] offer alternative methods for supporting CORBA object invocations in the wireless environment. Mobile DCE [37] is another early platform that successfully demonstrated that remote procedure calls could be utilised by mobile applications. However, the synchronous communication paradigm assumes a permanent connectivity that cannot be guaranteed in most mobile computing settings. Adaptations of traditional object-oriented middleware to mobile scenarios are therefore usually targeted to nomadic settings where cell hand-overs allow mobile devices to roam while being connected.

Although extensions to established platforms have illustrated that they can be utilised in wireless networks, they do not support the most natural interaction paradigm for this setting. The characteristics of wireless communication media (e.g., low and variable bandwidth, frequent disconnections, etc.) favour a decoupled and opportunistic communication paradigm: decoupled in the sense that computation proceeds even in the presence of disconnections, and opportunistic as it exploits connectivity whenever it becomes available. Therefore, middleware that supports asynchronous communication has been proposed as being more suitable for developing mobile applications. These generally take the form of publish-subscribe platforms and tuple spaces. Notably, however, Rover [20] was an initial platform in this area that implemented an asynchronous remote procedure call paradigm.

Within the publish-subscribe paradigm, interaction takes the form of event notification; namely, consumers subscribe to events they are interested in and are notified when they are published. Logically, the two parties do not have to be connected simultaneously to interact, therefore, suiting the mobile environment. Examples of these are Elvin [38], SIENA [8] and iBus [42]. However, these platforms were designed specifically for fixed networks and do not take into account the dynamic connection of mobile hosts. This has enforced the emergence of some preliminary solutions. For example, Elvin has been extended to incorporate proxy servers to support the persistence of events, so that clients who disconnect repeatedly do not lose events. Nonetheless, it requires that clients connect to the same proxy, which cannot be guaranteed in mobile networks. Another alternative is JEDI [13], which includes a dynamic tree of dispatchers for ensuring publish-subscribe information is retained as members connect and reconnect. Both techniques rely on centralised entities holding event information, which is not suitable for mobile applications based in ad-hoc wireless networks.

Although not initially designed for this purpose (their origins go back to Linda [18], a coordination language for concurrent programming), tuple space systems have been shown to provide many useful facilities for communication in wireless settings. Tuple spaces act as a repository of data structures called tuples that processes can concurrently access. Communication is de-coupled in both time and space: senders and receivers do not need to be available at the same time, because tuples have their own life span, independent of the process that generated them, and mutual knowledge of their location is not necessary for data exchange, as the tuple space looks like a globally shared data space, regardless of machine or platform boundaries. These forms of decoupling assume enormous importance in a mobile setting, where the parties involved in communication change dynamically due to their migration or connectivity patterns. Examples include, Lime [31], TSpaces [45] and L^2imbo [14]. With the exception of L^2imbo, these systems do not address the issue of dynamic adaptation of middleware to changing context.

The range of middleware types described illustrates the level of middleware heterogeneity that occurs within the mobile domain (i.e., the platforms that mobile services are advertised and implemented by, change as the device roams). As previously stated, reflection does not assume or require any particular communication paradigm. A reflective middleware can therefore provide any communication style (e.g., tuple space), or better, can provide different communication paradigms (e.g., RPCs, events, messages, etc.) among which applications can dynamically choose the one that best suits their current needs.

II.B. New Approaches

We hypothesise that the range of asynchronous middleware paradigms such as publish/subscribe and message oriented systems will play a key role in the success of mobile computing middleware. However, communication is not the only aspect that middleware should tackle: other important aspects such as data sharing and context awareness need to be addressed.

Middleware must support mobile applications to be aware of the context in which they are being used in order to enable applications to adapt to heterogeneity of hosts and networks, as well as variations in the user's environment. User context includes, but is not limited to: location, with varying accuracy depending on the positioning system used; relative location, such as proximity to printers and databases; device characteristics, such as processing power and input devices; physical environment, such as noise level and bandwidth; user activity, such as driving a car or sitting in a lecture theatre.

Context-aware computing is not a new computing paradigm; it was first proposed a decade ago [36] and many researchers have studied and developed systems that collect context information, and adapt to changes. In particular, location has attracted a lot of attention and many applications exploit location information to offer travellers directions, such as the Shopping Assistant [2] and CyberGuide [26]; to find out neighbouring devices and the services they provide, such as Teleporting [3]; to send messages to anyone in a specific area, such as a Conference Assistant [15]; and so on. Most of these systems interact directly with the underlying network OS to extract location information, process it, and present it in a convenient format to the user. One of their major limitations concerns the fact that they do not cope with heterogeneity of coordinate information, and therefore different versions have to be released that are able to interact with specific sensor technologies, such as the Global Positioning System (GPS) outdoors, and infrared and radio frequency indoors.

To enhance the development of location-based services and applications, and to reduce their development cycle, middleware systems have been built that integrate different positioning technologies by providing a common interface to the different positioning systems, thus solving heterogeneity issues. Examples include Oracle iASWE [33], Nexus [17], Alternis [1], SignalSoft [40], and CellPoint [10]. Interestingly enough, few contexts other than location have been actually used to date.

An exception can be found in [44], where an architecture is proposed that reports changes in the environment to interested applications. Changes are modelled as asynchronous events that are dispatched to environment aware applications through an event delivery mechanism that separates event detection from delivery, so that event producers and consumers are effectively decoupled. Events are hierarchically organised, and can be extended in order to deal with a variety of changes happening in the environment (e.g., different location, memory availability, network connectivity, etc.). However, changes in the environment affects only the application behaviour, and are not used to dynamically adapt middleware behaviour.

Another major issue addressed by mobile computing middleware is the support for disconnected operations and data-sharing. Systems like Odyssey [35] and Xmiddle [28] try to maximise availability of data,

giving users access to replicas; they differ in the way they ensure that replicas move towards eventual consistency, that is, in the mechanisms they provide to detect and resolve conflicts that naturally arise in mobile systems.

We assert that a reflective middleware may enhance the development of context-aware services and applications. Through reflection, application-specific information can be exploited, for example, to dynamically instruct middleware on which positioning system to use (if more than one is available). More generally, a reflective middleware would provide applications with context information that they can use to optimise (i.e., dynamically reconfigure) middleware, as well as their own behaviour, counter balancing, for example, the scarce resource availability.

III. Reflection

Reflection is a technique that first emerged in the programming language community to support the design of more open and extensible languages (e.g., see [21]). The key to the approach is to offer a meta-interface supporting the inspection and adaptation of the underlying virtual machine (the meta-level). The service offered by this interface is then referred to as the meta-object protocol. Access to this meta-level is provided through a process of reification. Reification effectively makes some aspect of the internal representation explicit and hence accessible from the application (the base-level). The opposite process is then absorption where some aspect of meta-system is altered or overridden.

The approach is nicely summarised by the following quote from Brian Cantwell Smith, the originator of the early work on reflection [41]:

> ”In as much as a computational process can be constructed to reason about an external world in virtue of comprising an ingredient process (interpreter) formally manipulating representations of that world, so too a computational process could be made to reason about itself in virtue of comprising an ingredient process (interpreter) formally manipulating representations of its own operations and structures”.

Reflection is now widely adopted in language design, as witnessed for example by the Java Core Reflection API and also the meta-object protocol defined in CLOS [21]. Reflection is also increasingly being applied to a variety of other areas including operating system design [46], concurrent languages [43], and increasingly distributed systems, as in [29] or [32]. Crucially, there is now a growing community working on the area of reflective middleware [4]. The main motivation for this research is to provide a principled (as opposed to ad-hoc) means of achieving openness (of the underlying middleware platform). For example, reflection can be used to inspect the internal behaviour of a platform (introspection). By exposing the underlying implementation, it becomes straightforward to insert additional behaviour to monitor the implementation, e.g., performance monitors, quality of service monitors, or accounting systems. Reflection can also be used to alter the internal behaviour of the middleware (adaptation). Examples include replacing or changing the implementation of the underlying transport protocol to operate more optimally over a wireless link, introducing an additional level of distribution transparency in a running computation (such as migration transparency), or inserting a filter component to reduce the bandwidth requirements of a communication stream.

In middleware platforms, two (complementary) styles of reflection have been used, namely structural and behavioural reflection.

- Structural reflection is concerned with the underlying structure of objects or components, e.g., in terms of interfaces supported. This is similar, for example, to the introspection features found in Java 1.2 and associated technologies, such as JavaBeans [30]. More advanced features may also be offered, such as the ability to adapt the structure of an object (e.g., to add new behaviour at run-time). Similarly, some systems provide architectural reflection, whereby the software architecture of the system can be reified and altered [5, 9], e.g. in terms of components and connectors. This can be applied to the very structure of the middleware platform itself, allowing the customisation of the architecture for the current environmental conditions. Finally, meta-data or context can be viewed as a form of structural reflection, providing additional (meta)information about the underlying system, e.g. physical location, current battery levels or performance of the network.

- Behavioural reflection is concerned with activity in the underlying system, e.g., in terms of the arrival and dispatching of invocations. Typical mechanisms provided include the use of interceptors that support the reification of the process

of invocation and the subsequent insertion of pre- or post- actions. Other systems provide similar capabilities through dynamic proxies [30]. Finally, some research has been carried out on providing access to underlying resources and associated resource management, e.g. through the reification of a set of logical tasks and enabling the customisation of resource allocation and management policies [16].

There has also been research on overall architectures that encompass the various forms of reflection [5].

IV. Projects

IV.A. CARISMA

In order to allow applications that execute on portable devices to adapt to frequent and unannounced changes in the environment, middleware systems must support context awareness. By context, we mean everything that can influence the behaviour of an application, from resources within the device, such as memory, battery power, screen size and processing power, to resources outside the physical device, such as bandwidth, network connection, location and other hosts within reach.

CARISMA, a project carried out at University College London, is a middleware that uses both structural and behavioural reflection to enable context-aware interactions between mobile applications. The middleware is in charge of maintaining a valid representation of the execution context, by directly interacting with the underlying network operating system. Depending on the current context, applications may require the middleware to behave in specific ways. For example, an image processing application may ask the middleware to display pictures in black and white when battery power is low, using full-size, full-colour pictures when battery power permits. In CARISMA, the middleware can be seen by applications as a dynamically customisable service provider. This customisation takes place by means of *application profiles*.

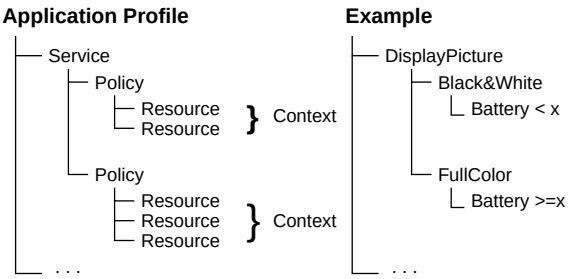

Figure 1: Structure of an application profile.

As shown in Figure 1, each application profile defines associations between the services that the middleware delivers, the policies that can be applied to deliver the services, and the context configurations that must hold in order for a policy to be applied. In the example above, an association is defined between the service 'DisplayPicture', the 'Black&White' policy, and a context where the resource 'Battery Power' is low, and another one between the same service 'DisplayPicture', the 'FullColour' policy, and a context where 'Battery Power' is high. Profiles are passed down to the middleware; each time a service is invoked, the middleware consults the profile of the application that requests it to determine which policy can be applied in the current context. Our model assumes that at each time the behaviour of the middleware with respect to a particular service is determined by one and only one policy, that is, a service cannot be delivered using a combination of different policies. If different policies need to be combined, a new name must be assigned to the combined policy, and this name must be used in the profile. For example, the display of an image can be done using a 'B&W-Low' policy, that is a combination of 'Black&White' and 'LowResolution'.

As both the needs of the user and the context change quite frequently, we cannot expect an application to fix its own profile once and for all at the time of installation and never change it after. We therefore need to provide the middleware with an initial profile, and then grant the application dynamic access to it. Here is where (structural) reflection comes into play. The middleware configuration with respect to current context is reified by means of application profiles, and altered through a reflective API that allows applications to read their own profile (introspection of middleware behaviour), and to dynamically modify the meta-data that is encoded in the profile (adaptation of middleware behaviour). If we consider once again the display picture example, an application may restrict the use of the 'FullColour' policy from a context where battery power is high to a context where both battery power and memory are high, due to a considerable drop in memory availability.

CARISMA uses also behavioural reflective techniques to dynamically customise resource allocation among mobile applications that are competing for limited resources, such as battery, memory and bandwidth [7]. A conflict exists when multiple policies, common to multiple application instances, can be applied at the same time to deliver a requested service, so that the middleware does not know which one to

apply (recall that we made the assumption that a service can be delivered using only one policy at a time). Different policies deliver different quality of service, and consume different amounts of (usually scarce) resources.

The resolution is based on an auction protocol where middleware plays the role of the auctioneer, applications are the bidders and policies are the goods they are competing for [7]. The main rule of the game is very simple: each bidder submits a single sealed bid, and the 'winner' is the bidder who bids highest. Bids are computed based on non-functional concerns, such as availability, security, performance. Reflection is used to allow applications (i.e., bidders) to dynamically change the set of non-functional requirements they care about; as a result, the same conflict may be resolved in different ways in different context, thus resulting in different middleware behaviours and different resources allocation.

We have implemented a prototype of our reflective middleware in Java, and developed a conferencing application and an instant messaging application on top of it. Application profiles have been encoded using the eXtensible Markup Language (XML). We chose to use XML as this meta-language enhances context-aware and user-driven interactions between middleware and applications, supporting a representation of information that is both easily manipulatable by machines and readily understandable by humans.

Performance evaluation of CARISMA is ongoing work. Preliminary results obtained running CARISMA on top of Compaq iPAQ PDAs running Linux, and connected using WaveLAN, have shown that it takes less than 200ms to start a local service, and less than 400ms for a remote service. We have also measured the overhead introduced by the conflict resolution mechanism; in case a conflict exists, the amount of time required to start a service increases by 10% for a local service, and of 20% for a remote one. In both cases, the there is no human perception of the delay as most of the time is taken by the loading and provisioning of the service. Further experiments will be run on multiple services, and compared with a system that does not use reflection. For a more complete discussion of CARISMA, please refer to [6, 7].

IV.B. ReMMoC

The ReMMoC project, being carried out at Lancaster University in collaboration with Lucent Technologies, is examining the use of reflection and component technology to overcome the problems of heterogeneous middleware technology in the mobile environment. Mobile applications and services are implemented upon a range of middleware platforms (e.g., RPC, message-oriented and event-based paradigms) and are advertised using different service discovery protocols. Therefore, developing mobile applications upon fixed middleware types is unsuitable; rather the middleware on the mobile device should be able to dynamically change its structure and behaviour so that it can discover and interoperate with all services available in the current environment.

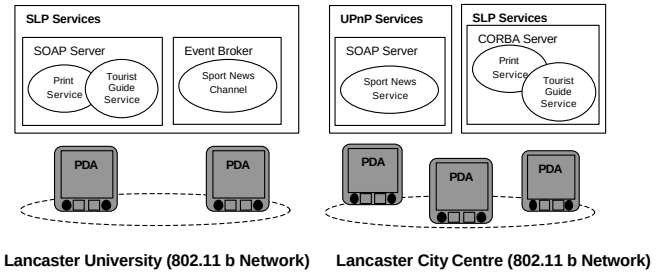

Figure 2: Heterogeneous mobile services.

To illustrate this, a typical mobile scenario is presented in Figure 2. Two possible locations in the session of a mobile user and the mobile services that can be interacted with via the given mobile device are illustrated. It can be seen that the same three mobile application services are available to the user in the different locations, but the platforms presenting them differ. For example, the sport news service is implemented as a publish-subscribe channel on the university campus network and as a SOAP service in the town centre. If fixed middleware were to be used, then two separate applications and middleware implementations would be needed on the device. Similarly, the print service and tourist guide service are implemented across different middleware types. However, this is not the only level of heterogeneity in the scenario; the services themselves must first be discovered by the mobile device before interaction can occur. Nevertheless, in this setting the service discovery technologies themselves are different, i.e., the services available to campus users are discoverable using the Service Location Protocol (SLP), while the services across the city centre can be found using both the Universal Plug and Play (UPnP) protocol and SLP. If the middleware can only perform one type of service discovery then it may miss some available resources and in the worst-case scenario find none of them.

ReMMoC is a middleware platform built upon the concepts of components and reflection as proposed in the OpenORB Project [5]. The platform is constructed out of component frameworks, which consist of a con-

figuration of components whose structure can be altered via a meta-object protocol. This meta-object protocol provides the ability to i) inspect and invoke the interfaces available within the framework, ii) view and alter the component architecture and, finally, iii) insert interceptors to dynamically add new behaviour.

The current platform consists of two key component frameworks for binding and service discovery, whose behaviour can be dynamically altered using the previously described reflective techniques. The binding framework can be re-configured between different types of interaction protocols (e.g., CORBA, SOAP, publish-subscribe, etc.). Similarly, the service discovery framework can change between one or more of the different service discovery technologies (e.g., SLP, Jini, UPnP, etc). Therefore, possible configurations could be SLP at one time, but changed to SLP and UPnP at another; this will allow services advertised by both technologies to be found. The information obtained from performing service discovery drives the reconfiguration of the binding framework to the appropriate interaction protocol. We are also currently exploring the use of XML to allow the application's requirements to be described independently of the underlying middleware technologies. ReMMoC will then base its adaptation upon meeting these requirements.

We have implemented the ReMMoC platform upon our own OpenCOM Light component model. OpenCOM Light has been developed for use on devices running the Windows CE Operating System, with the aim of creating a minimum component platform for devices of limited resources. The size of this component platform is currently 41 Kbytes and the components created upon it are on average 8 Kbytes in size. The platform is closely based on Microsoft COM but enhanced with richer reflective facilities. More information about the OpenCOM model can be found in [12]. Furthermore, we have implemented middleware functionality for SOAP, IIOP and publish/subscribe to support dynamic binding and we are working on SLP and UPnP implementations of service discovery behaviour.

IV.C. Relationship between the two platforms

The two platforms described previously are closely related, in that they use reflection to support two key requirements of mobile middleware. Firstly, ReMMoC presents the ability to develop applications independently from specific middleware technologies that may be encountered over time, by allowing the service discovery and communication implementation to be adapted dynamically. However, the control of adaptation needs to be based upon context information from both the environment and the device, in order for the middleware to offer the best level of service. The management of the underlying platform changes over time can be controlled by the properties offered by CARISMA. Therefore, a reflective framework, where policy and mechanism co-exist and can both be altered dynamically, provides support to mobile application developers to solve a host of problems.

V. Related Work

The principle of reflection has already been investigated by the middleware community during the past years, mainly to achieve flexibility and dynamic reconfigurability of the Object Request Broker (ORB) of CORBA. Examples include, but are not limited to, OpenCorba [24], dynamicTAO [23] and OpenORB [5]. However, the platforms developed to experiment with reflection were based on standard middleware implementations (except Open ORB as shown in section IV.B), and therefore targeted to a wired distributed environment.

Other approaches that specifically target the mobile setting include: Universally Interoperable Core (UIC [34]), a minimal reflective middleware for mobile devices where a pluggable set of components allows developers to specialise the middleware to suit different devices and environments, thus solving heterogeneity issues. The configuration can also be automatically updated both at compile-time and runtime. Personalities can be defined to have a client-side, server-side or combined behaviours. Personalities can also define the server type for an interaction (i.e., CORBA or Java RMI): single personalities allow the interaction with only one type, while multi personalities allow interaction with more than one type. In the case of multi personalities the middleware dynamically chooses the right interaction paradigm. The size of the core goes from 16KB for a client-side CORBA personality running on a Palm OS device to 37KB for a client/server CORBA personality running on a Windows CE device. UIC offers similar capabilities to the ReMMoC platform to overcome middleware heterogeneity. However, unlike ReMMoC, it concentrates on synchronous communication paradigms, less suited to mobile settings, and does not directly address the key property of heterogeneous service discovery.

2k [22] is an example of reflective, component-based meta-operating system based on the premise

"What You Need Is What You Get", that is, the system dynamically reconfigures itself to provide the functionality required by each particular entity. An entity may vary from users, to software components to devices. At any time, only a minimal set of components required for executing the user applications in the most efficient way is loaded, without carrying around non-utilised modules.

On top of this framework, Gaia [11] has been developed adding support for dynamic adaptation to context conditions. Gaia converts physical spaces and the ubiquitous computing devices they contain into active spaces, that is, programmable computing systems with well-defined behaviour and explicitly defined functionality. The goal of an active space is to provide a generic model that hides the complexity associated to its internal status and exports a standard programming interface independent of the underlying physical space. Gaia then adapts application requirements to the properties of its associated active space, without having the application to explicitly deal with the particular characteristics of every possible physical space where they can be executed.

VI. Conclusion and Future Work

We have argued why the principle of transparency that has driven the design of traditional middleware systems may not be optimal for mobile settings. Implementation details have to be made available to the above running applications to allow dynamic reconfiguration of the underlying system, based on different context conditions and varying application needs.

We propose reflection as a principle to accommodate the variety of requirements imposed by mobility. In order to provide the best service to the application, the middleware must be aware of its context and current behaviour and then alter according to environmental changes. The two properties of openness and adaptation provided by reflective middleware is particularly well suited to this. We have shown examples that support this hypothesis. In particular, we have illustrated two projects: CARISMA, where reflection is used to allow applications to dynamically customise middleware behaviour, and ReMMoC, where reflection is used to deal with heterogeneous middleware technology in mobile environments. We have shown that these are two complementary techniques that are closely related and that together they can support the development of mobile applications.

Some major drawbacks of the reflective approach are performance, integrity and security issues. When developing a reflective mobile middleware, the following questions must therefore be addressed carefully: how much are we willing to trade-off between flexibility and performance? What is the scope of changes when reconfiguring the system? And, finally, what happens if malicious programs enter into our system and use the reflective mechanism to misconfigure the middleware?

Despite being a powerful means to build mobile computing middleware, reflection alone is not enough. The devices are not able to know a-priori which components they are going to need in which situation, and which other devices they are going to encounter. Devices cannot load all the possible behaviours, due to memory constraints; moreover, new behaviours may be delivered from time to time to cope with unforeseen context configurations and new application needs. We propose a combination of reflection and different mobile code paradigms (e.g., code on demand, remote evaluation, etc.) to enhance the adaptability of mobile computing applications and introduce the required level of dynamism and flexibility into the middleware.

References

[1] Alternis S.A. Solutions for Location Data Mediation. http://www.alternis.fr/.

[2] A. Asthana and M. Cravattsand P. Krzyzanowski. An indoor wireless system for personalized shopping assistence. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 69–74, Santa Cruz, California, December 1994. IEEE Computer Society Press.

[3] 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.

[4] G. S. Blair, R. Campbell, F. Costa, and F. Kon. Proceedings of First International Workshop on Reflective Middleware (RM2000). http://www.comp.lancs.ac.uk/computing/RM2000/, April 2000.

[5] G. S. Blair, G. Coulson, A. Andersen, L. Blair, M. Clarke, F. Costa, H. Duran-Limon, T.Fitzpatrick, L. Johnston, R. Moreira, N. Parlavantzas, and K. Saikoski. The Design and Implementation of OpenORB v2. *IEEE DS Online,*

Special Issue on Reflective Middleware, 2(6), 2001.

[6] L. Capra, W. Emmerich, and C. Mascolo. Reflective Middleware Solutions for Context-Aware Applications. In *Proc. of REFLECTION 2001. The Third International Conference on Metalevel Architectures and Separation of Crosscutting Concerns*, volume 2192 of *Lecture Notes in Computer Science*, pages 126–133, Kyoto, Japan, September 2001. Springer Verlag.

[7] L. Capra, W. Emmerich, and C. Mascolo. A Micro-Economic Approach to Conflict Resolution in Mobile Computing. In *Proceedings of the 10th International Symposium on the Foundations of Software Engineering (FSE-10)*, Charleston, South Carolina, USA, November 2002. ACM Press. To appear.

[8] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[9] W. Cazzola, W. Savigni, A. Sosio, and F. Tisato. Rule-based Strategic Reflection: Observing and Modifying Behaviour at the Architectural Level. In *14th IEEE International Conference on Automated Software Engineering (ASE99)*, pages 263–266, Cocoa Beach, Florida, USA, October 1999.

[10] CellPoint, Inc. The CellPoint System. http://www.cellpt.com/thetechnology2.htm, 2000.

[11] R. Cerqueira, C. K. Hess, M. Roman, and R. H. Campbell. Gaia: A Development Infrastructure for Active Spaces. In *Workshop on Application Models and Programming Tools for Ubiquitous Computing (held in conjunction with the UBICOMP 2001)*, September 2001.

[12] M. Clarke, G. S. Blair, G. Coulson, and N. Parlavantzas. An Efficient Component Model for the Construction of Adaptive Middleware. In *IFIP / ACM International Conference on Distributed Systems Platforms (Middleware'2001)*, pages 160–178, Heidelberg, Germany, November 2001.

[13] G. Cugola, E. Di Nitto, and A. Fuggetta. Exploiting an event-based infrastructure to develop complex distributed systems. In *19th International Conference on Software Engineering (ICSE98)*, pages 261–270, 1998.

[14] N. Davies, A. Friday, S. Wade, and G. S.Blair. L2imbo: A Distributed Systems Platform for Mobile Computing. *ACM Mobile Networks and Applications (MONET), Special Issue on Protocols and Software Paradigms of Mobile Networks*, 3(2):143–156, 1998.

[15] A. K. Dey, M. Futakawa, D. Salber, and G.D. Abowd. The Conference Assistant: Combining Context-Awareness with Wearable Computing. In *Proc. of the 3^{rd} International Symposium on Wearable Computers (ISWC '99)*, pages 21–28, San Franfisco, California, October 1999. IEEE Computer Society Press.

[16] H. Duran-Limon and G. S. Blair. A Resource Management Framework for Adaptive Middleware. In *3rd IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC'2K)*, Newport Beach, California, USA, March 2000.

[17] 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.

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

[19] M. Haahr, R. Cunningham, and V. Cahill. Supporting CORBA Applications in a Mobile Environment (ALICE). In *5th Int. Conf. on Mobile Computing and Networking (MobiCom)*, pages 36–47. ACM Press, August 1999.

[20] A. Joseph, A. deLespinasse, J. Tauber, D. Gifford, and M. Kaashoek. Rover: A Toolkit for Mobile Information Access. In *15th Symposium on Operating Systems Principles (SOSP '95)*, pages 156–171, Colorado, U.S, December 1995.

[21] G. Kiczales, J. des Rivires, and D.G. Bobrow. The Art of the Metaobject Protocol. MIT Press, 1991.

[22] F. Kon, R.H. Campbell, M.D. Mickunas, K. Nahrstedt, and F.J. Ballesteros. 2k: A Distributed Operating System for Dynamic Hetero-

geneous Environments. In *9th IEEE International Symposium on High Performance Distributed Computing*, pages 201–210, Pittsburgh, August 2000. IEEE Computer Society Press.

[23] 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.

[24] T. Ledoux. OpenCorba: a Reflective Open Broker. In *Reflection'99*, volume 1616 of *Lecture Notes in Computer Science*, pages 197–214, Saint-Malo, France, 1999. Springer Verlag.

[25] M. Liljeberg, K. Raatikainen, M. Evans, S. Furnell, K. Maumon, E. Veldkamp, B. Wind, and S. Trigila. Using CORBA to Support Terminal Mobility. In *TINA '97*, 1997.

[26] S. Long, R. Kooper, G.D. Abowd, and C.G. Atkenson. Rapid prototyping of mobile context-aware applications: the Cyberguide case study. In *Proceedings of the Second Annual International Conference on Mobile Computing and Networking*, pages 97–107, White Plains, NY, November 1996. ACM Press.

[27] C. Mascolo, L. Capra, and W. Emmerich. Middleware for mobile computing (a survey). In *Tutorial Proceedings of the International Conference of Networking 2002*, Lecture Notes in Computer Science. Springer Verlag, May 2002. To appear.

[28] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. *Int. Journal on Personal and Wireless Communications*, 21(1):77–103, April 2002.

[29] J. McAffer. Meta-Level Architecture Support for Distributed Objects. In G. Kiczales, editor, *Reflection 96*, pages 39–62, San Francisco, 1996.

[30] Sun Microsystems. Java reflection. http://java.sun.com/j2se/1.3/docs/guide/reflection/index.html, 2002.

[31] A. L. Murphy, G. P. Picco, and G.-C. Roman. LIME: A Middleware for Physical and Logical Mobility. In *Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21)*, pages 524–533, May 2001.

[32] H. Okamura, Y. Ishikawa, and M. Tokoro. AL-1/D: A Distributed Programming System with Multi-Model Reflection Framework. In *Workshop on New Models for Software Architecture*, November 1992.

[33] Oracle Technology Network. Oracle9i Application Server Wireless. http://technet.oracle.com/products/iaswe/content.html, 2000.

[34] M. Roman, F. Kon, and R. Campbell. Reflective Middleware: From your Desk to your Hand. *IEEE Communications Surveys*, 2(5), 2001.

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

[36] B. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *Proc. of the Workshop on Mobile Computing Systems and Applications*, pages 85–90, Santa Cruz, CA, December 1994.

[37] A. Schill, B. Bellmann, W. Bohmak, and S. Kummel. System Support for Mobile Distributed Applications. In *2nd International Workshop on Services in Distributed and Networked Environments (SDNE)*, pages 124–131, Whistler, British Columbia, June 1995. IEEE Computer Society Press.

[38] W. Segall and D. Arnold. Elvin Has Left the Building: A Publish/Subscribe Notification Service with Quenching. In *Australian UNIX Users Group 97*, Brisbane, Australia, September 1997.

[39] J. Seitz, N. Davies, M. Ebner, and A. Friday. A CORBA-based Proxy Architecture for Mobile Multimedia Applications. In *2nd IFIP/IEEE International Conference on Management of Multimedia Networks and Services (MMNS '98)*, Versailles, France, November 1998.

[40] SignalSoft. Wireless Location services. http://www.signalsoftcorp.com/, 2000.

[41] B.C. Smith. Reflection and Semantics in a Procedural Programming Language. Phd thesis, MIT, January 1982.

[42] Softwired. iBus Mobile. http://www.softwired-inc.com/products/mobile/mobile.html, April 2002.

[43] T. Watanabe and A. Yonezawa. Reflection in an Object-Oriented Concurrent Language. In *OOPSLA 88*, volume 23, pages 306–415. ACM Press, 1988.

[44] G. Welling and B. Badrinath. An Architecture for Exporting Environment Awareness to Mobile Computing. *IEEE Transactions on Software Engineering*, 24(5):391–400, 1998.

[45] P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford. T Spaces. *IBM Systems Journal*, 37(3):454–474, 1998.

[46] Y. Yokote. The Apertos Reflective Operating System: The Concept and Its Implementation. In *OOPSLA92*, volume 28, pages 414–434. ACM Press, 1992.