Chapter 12: Principles of Mobile Computing Middleware

CECILIA MASCOLO, LICIA CAPRA and WOLFGANG EMMERICH
Dept. of Computer Science,
University College London,
Gower Street, London WC1E 6BT, UK.

1. Introduction

The popularity of wireless devices, such as laptop computers, mobile phones, personal digital assistants, smartcards, digital cameras etc. is rapidly increasing. Their computing capabilities are growing quickly, while their size is shrinking, allowing many of them to become more and more part of everyday life. These devices can be connected to wireless networks of increasing bandwidth, and software development kits are available that can be used by third parties to develop applications. The combined use of these technologies on personal devices enables people to access their personal information as well as public resources anytime and anywhere.

Applications on these types of devices, however, introduce challenging problems. Devices face temporary and unannounced loss of network connectivity when they move; they are usually engaged in rather short connection sessions; they need to discover other hosts in an ad-hoc manner; they are likely to have scarce resources, such as low battery power, slow CPU and little memory; they are required to react to frequent changes in the environment, such as change of location or context conditions, variability of network bandwidth, that will remain by orders of magnitude lower than in fixed networks.

When developing distributed applications, designers do not have to explicitly deal with problems related to distribution, such as heterogeneity, scalability, resource sharing and fault tolerance. *Middleware* developed upon network operating systems provides application designers with a higher level of abstraction, hiding the complexity introduced by distribution. Existing middleware technologies, such as transaction-oriented, message-oriented or object-oriented middleware have been built trying to hide distribution as much as possible, so that the system appears as a single integrated computing facility. In other words, distribution becomes *transparent*.

These technologies have been designed and are successfully used for stationary distributed systems. However, as it will become clearer in the following, some of the requirements introduced by mobility cannot be fulfilled by these existing traditional middleware. First, the interaction primitives, such as distributed transactions, object requests or remote procedure calls, assume a stable, high bandwidth and constant connection between components. Furthermore, synchronous point-to-point communication supported by object-oriented middleware systems, such as CORBA, requires a rendez-vous between the client asking for a service, and the server delivering that service. In mobile systems, on the contrary, unreachability is not exceptional and the connection may be unstable. Moreover, it is quite likely that client and server hosts are not connected at the same time, because of voluntary disconnections (e.g., to save battery power) or forced disconnection (e.g., loss of network coverage). Disconnection is treated as an occasional fault by many traditional middleware; techniques for data-sharing and replication that have been successfully adopted in traditional systems might not, therefore, be suitable, and new methodologies need to be explored.

Moreover, mobility introduces higher degrees of heterogeneity and dynamicity than traditional distributed systems. Mobile hosts might have to support different communication protocols,

according to the wireless links they are exploiting; look-up operations are more elaborate than in distributed systems, because of location variability. Middleware reconfiguration becomes essential in order to adapt to highly varying context conditions.

Finally, traditional middleware systems have been designed targeting devices with almost no resource limitations, especially in terms of battery power. On the contrary, even considering the improvements in the development of these technologies, resources of mobile devices will always be, by orders of magnitude, more constrained.

The aim of this chapter is to give an overview of how the requirements usually associated to distributed systems are affected by physical mobility issues. We discuss how traditional middleware, and middleware built targeting mobile systems can fulfil these requirements. We provide a framework and a classification of the most relevant literature in this area, highlighting goals that have been attained and goals that need to be pursued.

The chapter is structured as follows: Section 2 describes the main characteristics of mobile systems and highlights the many extents to which they differ from fixed distributed systems. Section 3 presents a reference model for middleware systems based on the requirements that need to be fulfilled; Sections 4 to 8 contain a detailed and comparative review of existing middleware for mobile systems, based on the model given. For every requirement category, we describe its main characteristics, illustrate some examples of solutions proposed to date, and highlight their strengths and limitations. Section 9 contains discussion and future directions of research in the area of middleware for mobile computing.

2 . Mobile Distributed Systems

In this section, we introduce a framework that we will use to highlight the similarities, but more importantly, the differences between fixed distributed systems and mobile systems.

2.1 Characterisation of Distributed Systems

A distributed system consists of a collection of components distributed over various computers (also called hosts) connected via a computer network. This definition of distributed system applies to both traditional and mobile systems. To understand the differences existing between the two, we now investigate three concepts hidden in the previous definition: the concept of *device*, of *network connection* and of *execution context*.

Type of Device: as a first basic distinction, devices in a fixed distributed system are stationary or *fixed*, while a mobile distributed system has at least some physically *mobile* devices. This is a key point: fixed devices vary from home PCs, to Unix workstations, to mainframes; mobile devices vary from personal digital assistants, to mobile phones, digital cameras and smartcards. While the former are generally powerful machines, with large amounts of memory and very fast processors, the latter have limited capabilities like slow CPU speed, little memory, low battery power and small screen size.

Type of Network Connection: fixed hosts are often *permanently* connected to the network through continuous high-bandwidth links. Disconnections are either explicitly performed for administrative reasons or are caused by unpredictable failures. These failures are treated as exceptions to the normal behaviour of the system. Such assumptions do not hold for mobile devices that connect to the Internet via wireless links. The performance of wireless networks (i.e., GSM, GPRS networks, satellite links, WaveLAN, HiperLAN, Bluetooth) may vary depending on

the protocols and technologies being used; reasonable bandwidth may be achieved, for instance, if the hosts are within reach of a few (hundreds) meters from their base station, and if they are few in number in the same base station cell. In some of the technologies, all different hosts in a cell share the bandwidth (i.e., if they grow in number, the quality of service rapidly drops). Moreover, if a device moves to an area with no coverage or with high interference, bandwidth may suddenly drop to zero and the connection may be lost. Unpredictable disconnections cannot be considered as an exception any longer, but they rather become part of normal wireless communication. Either because of failures or because of explicit disconnections, the network connection of mobile distributed systems is typically *intermittent*.

Type of Execution Context: with context, we mean everything that can influence the behaviour of an application; this includes resources internal to the device, like amount of memory or screen size, and external resources, like bandwidth, quality of the network connection, location or hosts (or services) in the proximity. In a fixed distributed environment, context is more or less *static*: bandwidth is high and stable, location almost never changes, hosts can be added, deleted or moved, but the frequency at which this happens is by orders of magnitude lower than in mobile settings. Services may change as well, but the discovery of available services is easily performed by forcing service providers to register with a well-known location service such as LDAP or DNS. Context is extremely *dynamic* in mobile systems. Hosts may come and leave generally much more rapidly. Service lookup is more complex in the mobile scenario, especially when the fixed infrastructure is completely missing, as for ad-hoc systems (Section 2.4). Broadcasting is the usual way of implementing service advertisement; however, this has to be carefully engineered in order to save the limited resources available (e.g., sending and receiving is power consuming), and to avoid flooding the network with messages. Location is no longer fixed: the size of wireless devices has shrunk to the point that most of them can be carried in a pocket and moved around easily. Depending on location and mobility, bandwidth and quality of the network connection may vary greatly.

2.2 Traditional Distributed Systems

According to the framework previously described, traditional distributed systems are a collection of fixed hosts, permanently connected to the network via high-bandwidth and stable links, executing in a static environment. The distributed components running on these hosts need to interact with each other in order, for example, to exchange data or to access services. To facilitate interactions, the following requirements have to be guaranteed [8][12]:

- **Fault-tolerance**: the ability to recover from faults without halting the whole system. Faults happen because of hardware or software failures (e.g., software errors, ageing hardware, etc.), and distributed components must continue to operate even if other components they rely on have failed.
- **Openness**: the possibility to extend and modify the system easily, for example, to respond to changed functional requirements. Any real distributed system will evolve during its lifetime. The system needs to have a stable architecture so that new components can be easily integrated while preserving previous investments.
- **Heterogeneity**: it calls for integration of components written using different programming languages, running on different operating systems, executing on different hardware platforms. In a distributed system, heterogeneity is almost unavoidable, as different components may require different implementation technologies.
- **Scalability**: the ability to accommodate a higher load at some time in the future. The load can be measured using many different parameters, such as, for instance, the maximum

number of concurrent users, the number of transactions executed in a time unit, the data volume that has to be handled.

- **Resource sharing**: in a distributed system, hardware and software resources (e.g., a printer, a database, etc.), are shared among the different users of the system; some form of access control of the shared resources is necessary in order to grant access to authorised users of the system only.

2.3 Mobile Nomadic Systems

Nomadic systems can be considered a compromise between totally fixed and totally mobile systems. They are based on a core of fixed routers, switches and hosts; at the periphery of this fixed network, base stations with wireless communication capabilities control message traffic to and from dynamic configurations of mobile hosts.

Let us consider the requirements isolated for distributed systems and see how the characteristics of nomadic systems (i.e., mobile devices, dynamic context, permanent connectivity) influence them.

- **Fault-tolerance**: in most of the existing nomadic systems, disconnection is still treated as an occasional fault and the general pattern of communication used by most of the applications relies on a continuous connection. Nomadic applications that require support for disconnected operations exist, due to, for example, frequent bandwidth drops or high connection costs. In these cases, devices should be able to rely on cached data, in order to perform some off-line computation.
- **Openness**: mobile nomadic systems are characterised by rapidly varying execution context, both in terms of available resources, and of devices and services in reach while moving. Adaptation and dynamic reconfiguration are called for, in order to react to context changes, and to interact with newly discovered services.
- **Heterogeneity**: the issue of heterogeneity is even more serious than in traditional distributed systems. Heterogeneity of hardware platforms, of programming languages, and of operating systems still exists. However, mobility seems to push the need for heterogeneity handling further. In a nomadic setting, for example, it is likely for a device to be relocated into an environment with different conditions.
- **Scalability**: the ability to accommodate an increasing number of services and hosts, while maintaining a reasonable quality of service, is still an issue in nomadic systems. Services are often residing on the core network; the main concern in terms of scalability is the ability to deal with the limitations of the wireless links at the periphery of the network.
- **Resource sharing**: as in traditional systems, the services and resources of the network are shared. This implies that some monitoring on the access needs to be guaranteed, both in terms of authentication and concurrency. Having wireless links implies further weakness in terms of security. Concurrency in traditional systems is usually solved using transaction mechanisms; however, these are quite heavy and usually rely on a permanent and stable connection.

2.4 Mobile Ad-Hoc Systems

Mobile ad-hoc (or simply ad-hoc) distributed systems consist of a set of mobile hosts, connected to the network through wireless links. They differ from traditional and nomadic distributed

systems in that they have no fixed infrastructure: mobile hosts can isolate themselves completely and groups may evolve independently. Connectivity may be asymmetric or symmetric depending, for instance, on the radio frequency of the transmission used by the hosts. Radio connectivity is, by default, not transitive. However, ad-hoc routing protocols have been defined [24] in order to overcome this limitation and allow routing of packets through mobile hosts. Pure ad-hoc networks have encountered so far limited applications that range from small ad-hoc groups to share information in meetings for a short time, to military applications on battlefields, and discovery or emergency networks in disaster areas.

The requirements discussed for distributed and nomadic systems still hold in this kind of networks, even though they are further complicated.

- **Fault-tolerance**: replication usually permits applications to cope with fault tolerance in common distributed and, to some extent, even nomadic system. However, in ad-hoc networks, the lack of infrastructure imposes that no hosts acting as servers can be assumed. The distinction between the role of servers and clients in mobile ad-hoc networks is in fact blurred.
- **Heterogeneity**: heterogeneity issues for ad-hoc networks are much the same as the ones we have described for nomadic networks. Because of the lack of a fixed infrastructure, it is more likely that hosts equipped with different network adaptors meet and are willing to communicate.
- **Openness**: the fact that no servers are available on a core network implies that new behaviours cannot be downloaded from a specific and known location, but need to be retrieved, if at all existing. Moreover, due to higher dynamicity than in fixed or nomadic networks, the chances that a host will come into contact with a service never met before, and for which it has no specific interaction behaviour, or that it will encounter a new environment, for which new functionalities are needed, increase.
- **Scalability**: coordination of many hosts in a completely unstructured environment is very challenging. Decentralised solutions for service discovery and advertisement need to be put in place and exploited in order to obtain the best results.
- **Resource-sharing**: again, the lack of structure implies that security is much more difficult to obtain. The existence of specific locations to register authentication data of hosts cannot be assumed; links are wireless, allowing for more risks of eves-dropping. Concurrency can be controlled using transactions; however, as in nomadic systems, this becomes more difficult due to mobility, scarce resources and connectivity. Moreover, while in nomadic systems core hosts could register the transaction session while a host is handing off a cell, in ad-hoc networks no such infrastructure can be assumed, and a host can just go out of range in the middle of a transaction.

In between nomadic and ad-hoc types of network, there is a large range of other network solutions, which adopt aspects of both. We believe these heterogeneous networks, where fixed components interact with ad-hoc areas, and where different connectivity technologies are used, are going to be the networks of the future.

3. Middleware Systems: a Reference Model

Building distributed applications, either mobile or fixed, on top of the network layer is extremely tedious and error-prone. Application developers have to explicitly deal with all the requirements listed in the previous section, such as heterogeneity and fault-tolerance, and this complicates

considerably the development and maintenance of an application. Given the novelties of mobile systems and the lack of adequate development support, some of the developed systems for mobile environments adopted the radical approach of not having a middleware but rather rely on the application to handle all the services and deal with the requirements, often using a context-aware approach that allows adaptation to changing context [7]. Sun provides J2ME (Java Micro Edition), which is a basic JVM and development package targeting mobile devices. Microsoft provides the .Net Compact Framework, which also has support for XML data and web services connectivity. However this approach is not sufficient, as it completely relies on application designers to address most of the requirements middleware should provide, and, as we have said, these requirements are difficult to meet and even further complicated by mobility.

During the past years, middleware technologies for distributed systems have been built and successfully used in industry. However, as we argued before, although object-oriented middleware have been very successful in fixed environments, these systems might not be suitable in a mobile setting. Researchers have both adapted traditional middleware for use in mobile setting, and have been, and are, actively working to design new middleware targeted to the mobile setting (Sections 4 to 8).

Before entering into the details of specific systems and solutions, we provide a general overview of middleware for fixed and mobile distributed systems based on this model.

3.1 Middleware for Fixed Distributed Systems

Middleware for fixed systems can be mainly described as resource-consuming systems that hide most of the details of distribution from application designers. Considering the reference model introduced before, middleware for fixed distributed systems can be further characterised as follows:

- **Fault-tolerance**: disconnection of hosts is considered infrequent and limited in time. Middleware treats disconnection as an exception and transparently recovers from the fault. Connectivity among the hosts is considered stable and synchronous communication paradigms are mostly used, even if some middleware offer support for asynchronous communication. Replication techniques are used in order to cope with faults and to recover data. Given the stability of the connection among the hosts, and the quite reliable knowledge of the topology of the network, servers for replication of data are usually established. This has implications on both scalability, as replicas distribute access load among different nodes, and on resource-sharing, as changes to the copies need to be synchronised.
- **Heterogeneity**: different technologies are used to implement servers, and different hardware components are deployed. Middleware for fixed distribute systems hides these differences through a set of primitives for communication that are independent from the underlying software and hardware.
- **Openness**: systems need to adapt to changes and to allow for new functionalities to be introduced. Middleware for distributed systems fulfil this requirement by defining specific interfaces that allow dynamic discovery of new functionalities, so that components can easily be integrated.
- **Scalability**: the ability to accommodate an increasing number of users and services, is critical in current distributed systems. Middleware tackles scalability issues mainly by replication of services and through hierarchical structuring of (replicated) lookup services. Transparent relocation of services is also used for scalability purposes.

- **Resource sharing**: transaction and security services are often a crucial part of existing middleware. Transaction servers are used to ensure the integrity of the data in a distributed setting, while authentication mechanisms are put in place to ensure that only trusted users and hosts are accessing the services.

3.2 Middleware for Mobile Nomadic and Ad-hoc Systems

As we have seen, nomadic systems share some characteristics with traditional distributed systems. However, the introduction of mobility imposes new constraints to the middleware that can be developed for these systems. Mobile ad-hoc systems are characterised by a complete lack of infrastructure. Mobility in this setting complicate matters further. The development of middleware for these systems is at a very early stage. With regard to the reference model, the following holds:

- **Fault-tolerance**: disconnections are frequent in nomadic and, especially, in ad-hoc systems; moreover, the connection is often unstable and/or expensive. The synchronous mechanisms for communication adopted in traditional middleware seem to fail in delivering the right service. Asynchronous communication seems to fit better in mobile scenarios. Mixed solutions that adopt synchronous communication integrated with caching and buffering techniques also have some space in the mobile middleware arena. The use of data replication techniques and caching assumes a crucial importance as the ability to cope with disconnections (both faulty and volunteer) is much more essential than in traditional systems. Off-line operations should be allowed, and data should be available on the device even if not connected to the server. Different techniques have been devised to allow replication and consistency of copies, and they will be described in the following sections.
- **Heterogeneity**: mobility causes devices to encounter highly dynamic context and heterogeneous environment. Different behaviours are required to allow devices to interact in these situations; however, due to resource limitations, only middleware with small footprints should be used. In order to cope with this discrepancy, reconfigurable middleware systems based on a compositional approach have been developed.
- **Openness**: the fact that the hosts are mobile means that they are very likely to change context and meet new services. Adaptability of the middleware to new conditions and new context is essential and may imply the use of techniques to do application dependent adaptation.
- **Scalability**: as in traditional systems, the ability to accommodate new hosts and services is essential. In nomadic settings, services may be replicated on the core network in a strategic way, so as to offer better quality of service to devices that are clustered in some areas. In order to offer good quality of service in crowded environments, some middleware have devised a leasing system, where resources are temporarily assigned to "users" and then withdrawn and reassigned, allowing a better resource allocation scheme that fits highly dynamic environments. In ad-hoc networks, service discovery should be very decentralised, as no central discovery service can be put in place.
- **Resource-sharing**: security in mobile systems is still a very immature field. Nomadic system security relies on authentication happening on core network servers, like in traditional systems. Mobile ad-hoc network security is more difficult to achieve, again due to the lack of fixed infrastructure, and the existing middleware do not offer much support in this sense. Transactions in nomadic environments are supported. However, different protocols than the ones used in traditional systems are used, in order to take into account mobility aspects. In ad-hoc settings, the absence of a core network does not allow complex transactions to take place. Research in how to obtain resource integrity

while having concurrent access is focusing on replication and weak consistency techniques.

Some middleware specifically targeting the needs of mobile computing have been recently devised [27]; assumptions such as scarce resources, and fluctuating connectivity have been made in order to reach light-weight solutions. Some of the approaches, however, only target a sub-set of the requirements listed before.

4. Fault Tolerance

Fault tolerance has been essential in the adoption of middleware for traditional distributed systems. However, fault tolerance in mobile systems needs to be handled slightly differently. In particular, two aspects of fault tolerance have been investigated by middleware researchers: connectivity and data sharing. We treat the two issues separately and show examples of mobile middleware that offer support for them.

4.1 Connectivity

Given the potential instability of wireless links, the possible low bandwidth or high costs, the frequent disconnections and the mobility involved, middleware for mobile systems should be able to provide support for fault tolerance in terms of connectivity. Some adaptation of traditional middleware to mobile systems have been attempted, allowing for a more flexible way of treating connectivity faults, based on caching and buffering.

Traditional Middleware applied in Mobile Computing. Object-oriented middleware has been adapted to mobile settings, mainly to make mobile devices inter-operable with existing fixed networks in a nomadic setting. The main challenge in this direction is in terms of software size and protocol suitability. IIOP (i.e., the Internet Inter-ORB Protocol) is the essential part of CORBA that is needed to allow communication among devices. IIOP defines the minimum protocol necessary to transfer invocations between ORBs. IIOP has been successfully ported to mobile setting and used as a minimal ORB for mobile devices. In ALICE [14] hand-helds with Windows CE and GSM network adaptors have been used to provide support for client-server architectures in nomadic environments. An adaptation of IIOP specifically for mobile (i.e., LW-IOP, Light-weight Inter-Orb Protocol) has been devised in the DOLMEN project: unsent data is cached and an acknowledgement scheme is used to face wireless medium unreliability. Actual names of machines are translated dynamically through a name server, which maintains up-to-date information of the host location. In [26] CORBA and IIOP are used together with the WAP (Wireless Access Protocol) stack in order to facilitate the use of CORBA services on a fixed network to mobile devices connected through WAP and a gateway. IIOP is used to achieve message exchange.

In general, the synchronous connectivity paradigm typical of the systems discussed above assumes a permanent connection that cannot be taken for granted in many mobile computing scenarios. These systems therefore usually target nomadic settings, where hand-offs allow mobile devices to roam while being connected. Only minimal support for disconnection is introduced.

Semi-asynchronous communication paradigms have also been investigated. RPC based middleware has been enhanced with queueing delaying or buffering capabilities in order to cope with intermittent connections. Examples of these behaviours are Rover [17] and Mobile DCE [33]. Totally asynchronous paradigms have been adapted too. An implementation of the message oriented middleware JMS (Java Messaging Server) has recently been released [34]. It supports

both point to point and publish/subscribe communication models, that is, a device can either communicate with a single other host (through its queue), or register for a topic and be notified of all the messages sent to that topic. We believe that the use of publish/subscribe and message oriented systems will be taken further as they offer an asynchronous communication mechanism that naturally fits into the mobile setting, providing support for disconnected operations.

Implementations of publish/subscribe systems, which still target fixed distributed systems, such as CORBA Component Model, begin to appear on the market [16]. It will probably not take long before this begins to be applied to nomadic settings.

Tuple Space-based Middleware. The characteristics of wireless communication media favour a decoupled and opportunistic style of communication: decoupled in the sense that computation proceeds even in presence of disconnections, and opportunistic as it exploits connectivity whenever it becomes available. Some attempts based on events [37], or queues (Rover or Mobile JMS) have been devised. Another asynchronous and decoupled communication paradigm has also been isolated as effective in mobile settings: although not initially designed for this purpose (their origins go back to Linda [13], a coordination language for concurrent programming), tuple space systems have been shown to provide many useful facilities for communication in wireless settings. In Linda, a tuple space is a globally shared, associatively addressed memory space used by processes to communicate. It acts as a repository (in particular a multi-set) of data structures called tuples that can be seen as vector of typed values. Tuples constitute the basic elements of a tuple space systems; they are created by a process and placed in the tuple space using a *write* primitive, and they can be accessed concurrently by several processes using *read* and *take* primitives, both of which are blocking (even if non-blocking versions can be provided). Tuples are anonymous, thus their selection takes place through pattern matching on the tuple contents. 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. However, a traditional tuple space implementation is not enough. There are basic questions that need to be answered: how is the globally shared data space presented to mobile hosts? How is it made persistent? The solutions developed to date basically differ depending on the answers they give to the above questions.

We now review a tuple-space middleware that have been devised for mobile computing applications: Lime [21]. Others exist such as Tspaces [38] or JavaSpaces. L2imbo [9] is a tuple space system where quality-of-service related aspects are added.

Lime. In Lime [21], the shift from a fixed context to a dynamically changing one is accomplished by breaking up the Linda tuple space into many tuple spaces, each permanently associated to a mobile unit, and by introducing rules for transient sharing of the individual tuple spaces based on connectivity.

Each mobile unit has access to an *interface tuple space* (ITS) that is permanently and exclusively attached to that unit and transferred along with it when movement occurs. Each ITS contains tuples that the unit wishes to share with others and it represents the only context accessible to the unit when it is alone. Access to the ITS takes place using conventional Linda primitives, whose

semantics is basically unaffected. However, the content of the ITS (i.e., the set of tuples that can be accessed through the ITS) is dynamically recomputed in such a way that it looks like the result of the merging of the ITSs of other mobile units currently connected. Upon arrival of a new mobile unit, the content perceived by each mobile unit through its ITS is recomputed taking the content of the new mobile unit into account. This operation is called engagement of tuple spaces; the opposite operation, performed on departure of a mobile unit, is called disengagement. The tuple space that can be accessed through the ITS of a mobile unit is therefore shared by construction and transient because its content changes according to the movement of mobile units.

4.2 Data-sharing

Mobility complicates the way data and services are shared among the hosts. In particular, the limitations of the resources on the device, and the unstable connectivity patterns make this task more difficult. Support for disconnected operations and data-sharing have been regarded as key points by middleware such as Coda [31], Bayou [35] and Xmiddle [20]. They try to maximise availability of data and tolerance for disconnections, 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. Despite a proliferation of different, proprietary data synchronisation protocols for mobile devices, we still lack a single synchronisation standard, as most of these protocols are implemented only on a subset of devices and are able to access a small set of networked data. This represents a limitation for both end users, application developers, service providers and device manufacturers.

Xmiddle. Xmiddle [20] allows mobile hosts to share data when they are connected, or replicate the data and perform operations on them off-line when they are disconnected; reconciliation of data takes place once the host reconnects. Unlike tuple-space based systems, which store data in flat unstructured tuples, Xmiddle allows each device to store its data in a tree structure (represented as XML files). Trees allow sophisticated manipulations due to the different node levels, hierarchy among the nodes, and the relationships among the different elements, which could be defined.

When hosts get in touch with each other, they need to be able to interact. Xmiddle allows communication through sharing of trees. On each host, a set of possible access points for the private tree is defined; they essentially address branches of the tree that can be modified and read by peers. The size of these branches can vary from a single node to a complete tree; unlike systems such as Coda, where entire collections of files have to be replicated, the unit of replication can be easily tuned to accommodate different needs.
In order to share data, a host needs to explicitly *link* to another host's tree. As long as two hosts are connected, they can share and modify the information on each other's linked data trees. When disconnections occur, both explicit (e.g., to save battery power or to perform changes in isolation from other hosts) and implicit (e.g., due to movement of a host into an out of reach area), the disconnected hosts retain replicas of the trees they were sharing while connected, and continue to be able to access and modify the data.

Xmiddle addresses ad-hoc networks. No assumption is made about the existence of more powerful and trusted hosts, which should play the role of servers and on which a collection of data should be replicated in full.

5. Heterogeneity

The ability to allow integration of different hardware, and software servers and clients is important in traditional systems and becomes essential in mobile systems. In the recent years, we have seen a proliferation of different small and mobile hosts with all sorts of network connectivity capabilities, displays, and resource constraints. All these hosts need to be networked and to communicate with each other. Overcoming heterogeneity comes with costs in terms of computational load, which conflicts with the scarce availability of resources of mobile hosts. The IIOP CORBA implementation that has been ported on mobile (discussed in Section 4), allows integration of different software servers and clients. However, research went beyond this and there have been some attempts towards solving heterogeneity issues at a higher level, by allowing static and dynamic reconfiguration of the middleware components on the mobile host. As it may have been noticed, heterogeneity issues are strictly related to openness issues. The use of reconfiguration techniques for dynamic adaptation is in fact at the basis of both.

Universal Interoperable Core. UIC (Universally Interoperable Core) [35] is at the basis of the implementation of Gaia (Section 5); it is a minimal reflective middleware that targets mobile hosts. UIC is composed of pluggable set of components that allow developers to specialise the middleware targeting different devices and environments, thus solving heterogeneity issues. The configuration can also be automatically updated both at compile and run time. Personalities (i.e., configurations) can be defined to have a client-side, server-side or both behaviours. Personalities can also define with which server type to interact (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, for instance, 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.

6. Openness

In mobile systems, openness mainly refers to the ability of the system to dynamically adapt to context changes. Adaptation allows, for example, to optimise the system behaviour based on current resource availability; to choose the protocol suite (e.g., communication protocol, service discovery protocol, etc.) that better targets the current environment; to easily integrate new functionalities and behaviours into the systems, and so on.

In traditional distributed systems, the principle of *reflection* has been exploited to introduce more openness and flexibility into middleware platforms, as presented in Chapter 3. In a mobile setting, the need of adaptation is pressing, due the high dynamicity of context. This need is coupled with resource limitations on portable devices, that forbid the deployment of complex and heavy-weight middleware platforms. Reflection may help solving these needs: a middleware core with only a minimal set of functionalities can be installed on a device, and then, through reflection, the system can be reconfigured dynamically to adapt to context changes.

Reflection is not the only principle investigated towards the solution of this problem. Researchers in context-aware computing have studied and developed systems that collect context information, and adapt to changes, even without exploiting the principle of reflection. User's 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's activity, such as driving a car or sitting in a lecture theatre.

In particular, location has attracted a lot of attention and many examples exist of applications that exploit location information to: offer travellers directional guidance, such as the Shopping Assistant [3] and CyberGuide [19]; to find out neighbouring devices and the services they provide, to send advertisements depending on user's location, or to send messages to anyone in a specific area. 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 reduce their development cycle, middleware systems have been built that integrate different positioning technologies by providing a common interface to the different positioning systems. Examples include Oracle iASWE [22], and many others are coming out. We will review some of these systems when discussing heterogeneity issues.

Odyssey. The mostly application transparent approach adopted by Coda has been improved introducing context-awareness and application-dependent behaviours in Odyssey [30], and allowing the use of these approaches in mobile computing settings. Odyssey assumes that applications reside on mobile clients but access or update data stored on remote, more capable and trustworthy servers; once again the nomadic scenario is targeted.

Odyssey proposes a collaborative model of adaptation. The operating system, as the arbiter of shared resources, is in the best position to determine resource availability; however, the application is the only entity that can properly adapt to given context conditions, and must be allowed to specify adaptation policies. This collaborative model is called application-aware adaptation.

Although better suited to the mobile environment than its predecessor Coda, Odyssey suffers from some limitations: the data that can be moved across mobile hosts (i.e., a collection of files) may be too coarse-grained in a mobile setting, where hosts have limited amount of memory and connection is often expensive and of low quality.

Gaia [6] shares the idea of offering the ability to change the behaviour of the middleware and of the application based on knowledge about the changing context; they differ in the way they achieve this goal. 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.

7. Scalability

Scalability issues are related to the ability of accommodating large numbers of hosts, both in terms of, for instance, quality of service and discovery of services. The fact that mobile links are unstable, expensive and sometimes limited in terms of bandwidth implies that quality of service rapidly decreases when the number of hosts involved increases. There have been some attempts to use middleware in order to solve scalability issues related to quality of service and we will give some examples. Discovery of services when hosts are mobile is also challenging, especially in completely ad-hoc scenarios where no centralised discovery service can be used. Some of the

middleware already presented, and some that we are going to introduce, offer a solution to these issues.

7.1 Discovery

In traditional middleware systems, service discovery is provided using fixed name services, which every host knows the existence of. The more the network becomes dynamic, the more difficult service and host discovery becomes. Already in distributed peer-to-peer network [23], service discovery is more complex as hosts join and leave the overlay network very frequently. In nomadic systems, service discovery is still very similar to service discovery in traditional systems, where a fixed infrastructure containing all the information and the services is present. However, in terms of more ad-hoc or mixed systems, where services can be run on roaming hosts, discovery may become very complex and/or expensive.

Most of the ad-hoc systems encountered till now have their own discovery service. Lime and Xmiddle use a completely ad-hoc strategy where hosts continuously monitor their environment to check who is available and what they are offering. A trade-off between power and bandwidth consumption (i.e., broadcast) and discovery needs to be evaluated. Recently, some work on Lime for service advertisement and discovery has been devised [15]. Standard service discovery frameworks have appeared in the recent years: UPnP [36], Jini [2], and Salutation [Salutation Consortium, 1999]. UPnP stands for Universal Plug and Play and it is an open standard for transparently connecting appliances and services, which is adopted by the Microsoft operating systems. UPnP can work with different protocols such as TCP, SOAP, HTTP. Salutation is a general framework for service discovery, which is platform and OS independent. Jini is Java-based and dependent on the Java Virtual Machine. The purpose of these frameworks is to allow groups of hosts and software components to federate into a single, dynamic distributed system, enabling dynamic discovery of services inside the network federation.

Jini and JMatos. Jini [2] is a distributed system middleware based on the idea of federating groups of users and resources required by those users. Its main goal is to turn the network into a flexible, easily administered framework on which resources (both hardware devices and software programs) and services can be found, added and deleted by humans and computational clients. The most important concept within the Jini architecture is the service. A service is an entity that can be used by a person, a program or another service. Members of a Jini system federate in order to share access to services. Services can be found and resolved using a lookup service that maps interfaces indicating the functionality provided by a service to sets of objects that implement that service. The lookup service acts as the central marketplace for offering and finding services by members of the federation. A service is added to a lookup service by a pair of protocols called *discovery* and *join*: the new service provider locates an appropriate lookup service by using the first protocol, and then it joins it, using the second one. A distributed security model is put in place in order to give access to resources only to authorised users.

Jini assumes the existence of a fixed infrastructure, which provides mechanisms for hosts, services and users to join and detach from a network in an easy, natural, often automatic, manner. It relies on the existence of a network of reasonable speed connecting Jini technology-enabled hosts. However the large footprint of Jini (3 Mbytes), mainly due to the use of Java RMI, prevents the use of Jini on smaller devices such as iPAQs or PDAs. In this direction, Psinaptic JMatos [25] has been developed, complying with the Jini Specification. JMatos does not rely on Java RMI for messaging and has a footprint of just 100 KB.

7.2 Quality of Service

In the existing examples of use of traditional middleware in the mobile setting, the focus is on the provision of services from a back-bone network to a set of mobile hosts: the main concerns in this scenarios are connectivity and message exchange. In case of a less structured network, or in case services must be provided by mobile hosts, traditional middleware paradigms seem to be less suitable and a new set of strategies needs to be used. The importance of monitoring the condition of the environment, and adaptation to application needs, maybe through communication of context information to the upper layers, becomes vital to achieve reasonable quality of service.

Given the highly dynamic environment and the scarce resources, quality of service provision presents higher challenges in mobile computing. Nevertheless, researchers have devised a number of interesting approaches to quality of service provision to mobile hosts [7]. Most of the time the hosts are considered terminal nodes and the clients of the service provision, and the network connectivity is assumed fluctuating but almost continuous (like in GSM settings).

Mobiware. Probably the most significant example of quality of service oriented middleware is Mobiware[1], which uses CORBA, IIOP and Java to allow service quality adaptation in mobile setting. In Mobiware, mobile hosts are seen as terminal nodes of the network and the main operations and services are developed on a core programmable network of routers and switches. Mobile hosts are connected to access points and can roam from one access point to another.

Mobiware mostly assumes a service provision scenario where mobile hosts are roaming but permanently connected, with fluctuating bandwidth. Even in the case of the ad-hoc broadband link, the host is supposed to receive the service provision from the core network through, the cellular links first, and then some ad-hoc hops. In more extreme scenarios, where links are all ad-hoc, these assumptions cannot be made and different middleware technologies need to be applied. One of the strength of Mobiware is the adaptation component to customise quality of service results.

8. Resource-sharing

Resource sharing is a very delicate and important aspect of mobile computing middleware. It involves two main issues that we are now going to tackle: transactions and security. Traditional middleware have successfully dealt with these issues for fixed distributed systems; however, those solutions seem to be cumbersome and inappropriate in a mobile scenario. Some new solutions for these issues have been proposed, especially in the area of nomadic system, where the core network helps in reusing traditional middleware solutions; however, the state of the art is very poor in terms of ad-hoc systems.

8.1 Transactions

We already introduced techniques for data sharing and consistency in Section 4. These techniques can, for some aspects, be considered as resource sharing techniques. However, given the tight link to connectivity we introduced them in that section. Those techniques are probably the best that can be achieved in mobile ad-hoc network where no infrastructure can support transactions. More elaborate techniques for resource sharing can be developed in nomadic systems where transaction servers can be located on the core network. For example, T-Spaces allows transactions to be managed on the server to guarantee consistent access to tuples. Protocols for dealing with transaction on nomadic systems have been developed. However, there is little novelty with respect to the traditional middleware transaction solutions. We now illustrate a couple of the most original examples; the ideas behind these approaches are quite similar to the ones presented while

describing Bayou and Coda. Jini (Section 7) also offers leases for resource sharing and Lime (Section 4) has an approach to transactions based on engagement and disengagements of tuple spaces. However, the following middleware is an example of a system more focused on the idea of transactions.

Kangaroo. The main idea behind Kangaroo[11] is that distributed transactions necessary for consistent resource sharing in mobile nomadic environments are subject to faults due to disconnections. In order to deal with these disconnections, Kangaroo models hopping transactions where the state of the started transaction is recorded at the base station where the host is connected. After disconnection, the host having recorded the last base station that offered connectivity, can resume the transaction using the log at the base station. Base stations contain agents which manage the hopping transactions. Kangaroo offers two modes of operation: compensating and split. In compensating mode, the failure of any sub-transaction at a base station causes the failure of the chain of all the transactions. In split mode, which is the default mode, the failure of a sub-transaction at a base station does not cause the failure of the sub-transactions committed after the previous hops. Kangaroo targets nomadic settings, where it offers interesting performance improvement, dealing with disconnections. However, given the fact that it completely relies on the existence of server side databases, Kangaroo does not seem to be the appropriate solution for ad-hoc systems.

8.2 Security

Security requirements for mobile networks are similar to those for traditional systems: authentication, confidentiality, integrity, non-repudiation, access control and availability. However, the mobility aspect introduced and the lack of infrastructure in the case of ad-hoc networks increase the number of challenges for the implementation and fulfilment of the above requirements. In nomadic systems, techniques already used in traditional systems can be applied, as the existence of servers where authentication data can be stored is assumed. However, given the involvement of wireless links, this has to be done in combination with some network level security (e.g., WEP, message encryption) to avoid eavesdropping. The limitation in terms of resources (e.g., battery power, processing power) also limits the kinds of encryption that can be used as this is power draining. In case of ad-hoc networks, the absence of any infrastructure makes things more challenging. Servers for authentication cannot be used, hosts should trust each other on the basis of simple and maybe application-specific information. Security requirement in mobile environments is highly related to some of the other requirements we discussed in this chapter. Most importantly, a host can be "attacked" by jamming his wireless connections to others. This problem is related to scalability and quality of service issues described in Section 7.

Middleware for nomadic systems including the fulfilment of security requirements are many. They all adopt the server-based authentication strategy adopted in traditional systems. In terms of ad-hoc networks, we have not seen very mature solutions put forward in this direction. All the middleware we have presented rely on trust of the hosts involved in the applications.

10. Conclusions

Mobile computing middleware research still faces many challenges that might not be solvable adapting traditional middleware techniques. We believe the future mobile networks will be heterogeneous in the sense that many different hosts will be available on the market, with possibly different operating systems and user interfaces. The network connectivity will also be heterogeneous even if an effort towards complete coverage through different connection technologies will be made. For these reasons, mobile computing middleware will have to adapt and be customisable in these different dimensions, both at start-up time (i.e., in case of adaptation

to different operating systems) and at run-time (i.e., in case of adaptation to different connection technologies).

We also believe application dependent information could play an important role in the adaptation of the behaviour of the middleware and in the trade-off between scarce resource availability and efficient service provision. In this direction, the effort of presentation of the information to the application, and the gathering of application dependent policies, is an important presentation layer issue that should be integrated in any mobile computing middleware.

Discovery of existing services is a key point in mobile systems, where the dynamicity of the system is, by orders of magnitude, higher than in traditional distributed systems. Recently, interesting research advances in peer-to-peer systems have focused on discovery issues that might be applicable, at least partially, to mobile settings. However, considerations on the variability of the connection, of the load and of the resources might be different for mobile scenarios. Furthermore, the integration of quality of service consideration into the service advertisement and discovery might enable some optimisation in the service provision.

Another direction of research concerns security. Portable devices are particularly exposed to security attacks as it is so easy to connect to a wireless link. Dynamic customisation techniques seem to worsen the situation. Reflection is a technique for accessing protected internal data structures and it could cause security problems if malicious programs break the protection mechanism and use the reflective capability to disclose, modify or delete data. Security is a major issue for any mobile computing application and therefore proper measures need to be included in the design of any mobile middleware system.

10 References

[1] Angin, O., Campbell, A., Kounavis, M., and Liao, R. 1998. The Mobiware Toolkit: Programmable Support for Adaptive Mobile Netwoking. In Personal Communications Magazine, Special Issue on Adapting to Network and Client Variability (August 1998). IEEE Computer Society Press.

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

[3] Asthana, A. and Krzyzanowski, M. C. P. 1994. An indoor wireless system for personalized shopping assistence. In Proceedings of IEEE Workshop on Mobile Computing Systems and Applications (Santa Cruz, California, Dec. 1994), pp. 69–74. IEEE Computer Society Press.

[4] Bennett, F., Richardson, T., and Harter, A. 1994. Teleporting - making applications mobile. In Proc. of the IEEE Workshop on Mobile Computing Systems and Applications (Santa Cruz, California, Dec. 1994), pp. 82–84. IEEE Computer Society Press.

[6] Cerqueira, R., Hess, C. K., Romn, M., and Campbell, R. H. 2001. 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) (Sept. 2001).

[7] Chalmers, D. and Sloman, M. 1999. A Survey of Quality of Service in Mobile Computing Environments. IEEE Communications Surveys Second Quarter, 2–10.

[8] Coulouris, G., Dollimore, J., and Kindberg, T. 2001. Distributed Systems – Concepts and Design, 3rd edition. Addison-Wesley.

[9] Davies, N., Friday, A., Wade, S., and Blair, G. 1998. L2imbo: A Distributed Systems Platform for Mobile Computing . ACM Mobile Networks and Applications. Special Issue on Protocols and Software Paradigms of Mobile Networks 3, 2.

[10] Demers, A., Petersen, K., Spreitzer, M., Terry, D., Theimer, M., and welch, B. 1994. The Bayou Architecture: Support for Data Sharing among Mobile Users. In Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications (Santa Cruz, California, Dec. 1994), pp. 2–7.
[11] Dunham, M. H., Helal, A., and Balakrishnan, S. 1997. A Mobile Transaction Model That Captures Both the Data and Movement Behavior. ACM/Baltzer Journal on Special Topics in Mobile Networks and Applications (MONET) 2, 149–162.
[12] Emmerich, W. 2000. Engineering Distributed Objects. John Wiley & Sons.
[13] Gelernter, D. 1985. Generative Communication in Linda. ACM Transactions on Programming Languages and Systems 7, 1, 80–112.
[14] Haahr, M., Cunningham, R., and Cahill, V. 1999. Supporting CORBA Applications in a Mobile Environment (ALICE). In 5th Int. Conf. on Mobile Computing and Networking (MobiCom) (August 1999). ACM Press.
[15] Handorean, R. and Roman, G.-C. 2002. Service Provision in Ad-hoc Networks. In F. Arbab and C. L. Talcott Eds., COORDINATION, Volume 2315 of Lecture Notes in Computer Science (2002), pp. 207–219. Springer.
[16] iCMG. 2002. K2: CORBA Component Server. http://www.componentworld.nu/.
[17] Joseph, A. D., Tauber, J. A., and Kaashoek, M. F. 1997. Mobile Computing with the Rover Toolkit. IEEE Transactions on Computers 46, 3.
[18] Kon, F., Rom´an, M., Liu, P., Mao, J., Yamane, T., aes, L. M., and Cambpell, R. 2000. Monitoring, Security, and Dynamic Configuration with the dynamicTAO Reflective ORB. In International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000) (New York, April 2000), pp. 121–143. ACM/IFIP.
[19] Long, S., Kooper, R., Abowd, G., and Atkenson, C. 1996. Rapid prototyping of mobile context-aware applications: the Cyberguide case study. In Proceedings of the Second Annual International Conference on Mobile Computing and Networking (White Plains, NY, Nov. 1996), pp. 97–107. ACM Press.
[20] Mascolo, C., Capra, L., Zachariadis, S., and Emmerich, W. 2002. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. Int. Journal on Personal and Wireless Communications 21, 1 (April).
[21] Murphy, A. L., Picco, G. P., and Roman, G.-C. 2001. Lime: A Middleware for Physical and Logical Mobility. In Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS-21) (May 2001).
[22] Oracle Technology Network. 2000. Oracle9i Application Server Wireless. http://technet.oracle.com/products/iaswe/content.html.
[23] Oram, A. 2001. Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly.
[24] Perkins, C. 2001. Ad-hoc Networking. Addison-Wesley.
[25] Psinaptic. 2001. JMatos. http://www.psinaptic.com/.
[26] Reinstorf, T., Ruggaber, R., Seitz, J., and Zitterbart, M. 2001. A WAP-based Session Layer Supporting Distributed Applications in Nomadic Environments. In Int. Conf on Middleware (Nov. 2001), pp. 56–76. Springer.
[27] Roman, G.-C., Murphy, A. L., and Picco, G. P. 2000. Software Engineering for Mobility: A Roadmap. In The Future of Software Engineering - 22nd Int. Conf. on Software Engineering (ICSE2000) (May 2000), pp. 243–258. ACM Press.
[28] Roman, M., Kon, F., and Campbell, R. 2001. Reflective Middleware: From your Desk to your Hand. IEEE Communications Surveys 2, 5.
[29] Salutation Consortium. 1999. Salutation. http://www.salutation.org/.
[30] Satyanarayanan, M. 1996. Mobile Information Access. IEEE Personal Communications 3, 1 (Feb.), 26–33.

[31] Satyanarayanan, M., Kistler, J., Kumar, P., Okasaki, M., Siegel, E., and Steere, D. 1990. Coda: A Highly Available File System for a Distributed Workstation Environment. IEEE Transactions on Computers 39, 4 (April), 447–459.
[32] Schilit, B., Adams, N., and Want, R. 1994. Context-Aware Computing Applications. In Proc. of the Workshop on Mobile Computing Systems and Applications (Santa Cruz, CA, Dec. 1994), pp. 85–90.
[33] Schill, A., Bellmann, W., and Kummel, S. 1995. System support for mobile distributed applications. IEEE Computer, 124–131.
[34] Softwired. 2002. iBus Mobile. http://www.softwired-inc.com/products/mobile/mobile.html.
Sun. JavaSpaces. 1998.
[35] Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M., and Hauser, C. 1995. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In Proceedings of the 15th ACM Symposium on Operating Ubi-core. 2001. Universally Interoperable Core. http://www.ubi-core.com.
[36] UPnP Forum. 1998. Universal Plug and Play. http://www.upnp.org/.
[37]Welling, G. and Badrinath, B. 1998. An Architecture for Exporting Environment Awareness to Mobile Computing. IEEE Transactions on Software Engineering 24, 5, 391–400.
[38] Wyckoff, P., McLaughry, S. W., Lehman, T. J., and Ford, D. A. 1998. T Spaces. IBM Systems Journal 37, 3, 454–474.