Chapter 7

Mobile Middleware: Definition and Motivations

Dario Bruneo, Antonio Puliafito,
and Marco Scarpa

CONTENTS

Basic Concepts

One of the main reasons for the wide and rapid spread of computer networks is the concept of *transparency*, traditionally embodied in the layered protocol stack [41]. Due to this simple but extremely powerful approach, users accessing a computer network are not aware (and do not want to be) of the technical details, protocols, and management issues. Usually, users want to run an application and get results without any knowledge of all the operations involved. If the application is a distributed application, very likely a connection must be established, followed by negotiation with regard to the most appropriate communication parameters and other complex activities. All of these activities are hidden to most network users.

Distributed Systems

With the increasing use of computer networks, the software architecture has changed radically. From a relatively simple architecture (*centralized system*) where all the software components are executed on a single machine, new software architectures (*distributed systems*) have been developed where software is organized into different modules distributed to different *elaboration nodes*, and data is exchanged by means of a communication network. In a centralized system, an application runs on a single node and constitutes a single process; the workstation is the only *active* component of the system because it hosts the application itself. Terminals share the resources of the workstation in such a way that different users can use the application. Terminals can even exist without a CPU, being equipped instead with a communication interface only for sending commands to the running application on the workstation. To summarize, in such a system the communication network is used to interconnect *stupid* nodes (*terminals*) with the *elaboration* node, as depicted in Figure 7.1.

In a distributed system an application is composed of more processes running on different nodes of the network. All the processes are cooperating closely, and they execute in parallel. So, a characteristic of distributed systems is that the processes do not share memory but instead rely on message exchange, thus introducing delay during the computation. The reference architecture changes, as depicted in Figure 7.2.

Because the network communication infrastructure has become very inexpensive over time, the computational power achieved by a distributed system is less expensive than that of an equivalent workstation. Moreover, the entire infrastructure is more manageable, as it is relatively simpler to increase the resources, to balance the workload, and to run parallel applications. In a distributed system, the reference programming paradigm

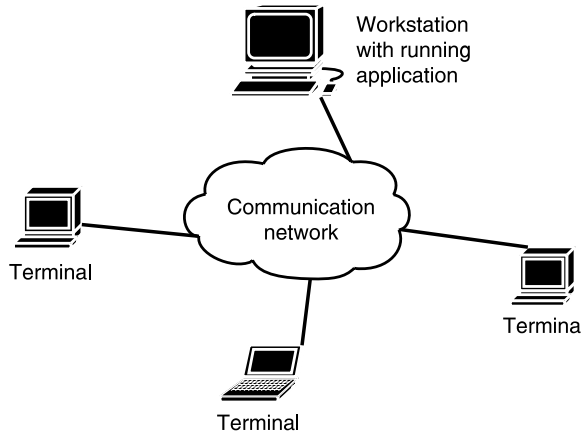

Figure 7.1 Example of a centralized system.

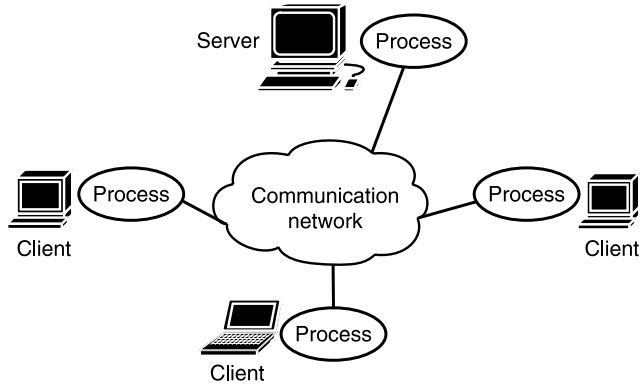

Figure 7.2 Example of a distributed system.

usually adopted is the *client–server* model. *Client* and *server* are distinct processes running on different nodes characterized by a well-defined interface. The server usually makes available some procedures for handling the data that are designed for responding to criteria of general effectiveness. The actual data processing is left to the server, where *ad hoc* procedures for the desired processing can be executed. The typical functional scheme is (Figure 7.3):

- The client asks the server for a service.
- The server performs the requested elaboration and sends the results back to the client.

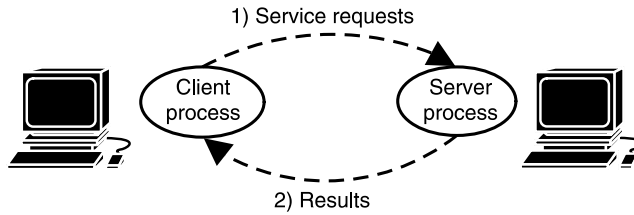

Figure 7.3 Client–server interaction.

Note that the distinction between *client* and *server* is based purely on function; in other words, a server could be a client of another server.

The Middleware Layer

Programming in a distributed system environment is a very tricky activity for developers, who must manage all the details related to the communication (e.g., addressing, error handling, data representation). This is due to the fact that developers use the low-level abstraction provided by the network operating system. To free developers from the expensive and time-consuming activity of resolving all the problems related to the network management, greater abstraction must be introduced. This is exactly the role of the *middleware* layer.

Middleware is a software layer between the operating system and the applications that provides a higher degree of abstraction in distributed programming. Using middleware, a programmer can develop an improved-quality distributed software by using the most appropriate, correct, and efficient solutions embedded in the middleware. In other words, utilizing middleware to build distributed systems frees developers from the implementation of low-level details related to the network, such as concurrency control, transaction management, and network communication, in such a way that they can focus on application requirements. Now consider the International Organization for Standardization (ISO)/Open Source Initiative (OSI) network reference model. Because middleware allows programmers to develop distributed systems as integrated computing facilities, it must address shortcomings of the network operating system; therefore, it implements the session and presentation layers of the ISO/OSI reference model [41], as depicted in Figure 7.4. In this environment, developers are able to request parameterized services from remote components and they can execute them without worrying about implementation of the session and presentation layers. Some examples of middleware successfully used thus far include OMG's CORBA™ [36], Microsoft's COM [8], SUN's Java Remote

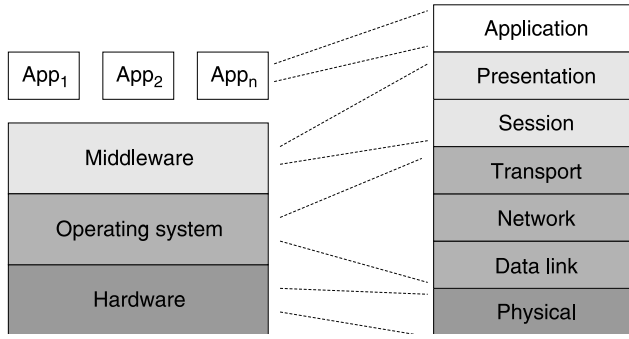

Figure 7.4 Software models and their relation with network models.

Method Invocation (RMI) [40], IBM's MQSeries™ [23], and remote procedure calls (RPCs), which were introduced by SUN in the 1980s [7].

As an example of middleware, we can take a look at RPC behavior. RPC middleware was created to give developers some kind of mechanism for accessing remote resources using just a local procedure call to hide all the details on the connection setup, marshal all the parameters, and hide all the problems related to the heterogeneity of different platforms. When the client performs a call, it is intercepted by the middleware, which runs the code for gathering the parameters, opening a socket, and so on (see Figure 7.5). The user is completely unaware of what happens. The middleware generates an appropriate message containing all the data related to that call and transmits it to the server, which is able to understand the request sent from the client and execute the procedure. When the

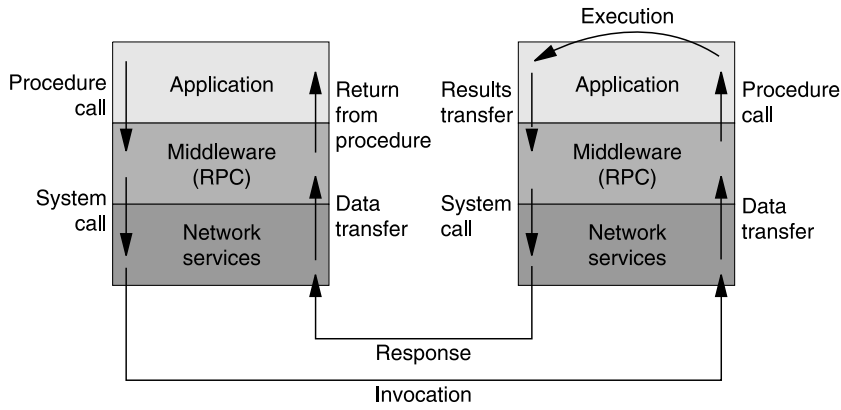

Figure 7.5 Remote procedure call (RPC).

results are ready, they are sent back to the client middleware, which returns to the application. The user perceives the process as being nothing more than a traditional local call.

Middleware Requirements for Fixed Distributed Systems

Middleware is a software layer that hides all the implementation details of the communication infrastructure to developers in such a way that they can overlook all the problems arising due to directly using network operating system primitives. To create this kind of abstraction, some requirements must be satisfied. It should be pointed out that some of these requirements must be altered when the middleware is to be used for mobile systems:

- *Communication* — First of all, the middleware has to guarantee the exchange of data among the nodes of a distributed system in such a way that the system appears to the user to be an integrated system. When two entities communicate, they exchange data as parameters of some kind of service; due to the heterogeneity of the nodes in the distributed system, the internal representation of data could be different, and this problem must be anticipated by the middleware. Moreover, all the parameters must be appropriately composed to be correctly exchanged; the middleware has to implement marshaling–unmarshaling operations.
- *Coordination* — The evolution of processes in distributed systems has to be controlled to achieve maximum cooperation among the different activities. Usually, most threads execute on the same host concurrently, and all of them must be synchronized with the remote one. Another issue is the fact that a distributed system could be viewed as a big repository of services; each service runs on a host, and it is available for invocation from other components of the system. In general, it is unknown when an invocation might come, and it is a waste of resources for the service to constantly execute; for these reasons, *activation* and *deactivation* must be provided by the middleware in order to start and stop the services when needed.
- *Reliability* — The network layers below the middleware ensure that communication errors and faults of the network are transparently recovered, but other kinds of errors can arise in the usual activities of a distributed application. As an example, nothing can be said about correct execution of a service or the order of the requests. This kind of reliability is implemented

directly by the middleware by using the *best effort*, *at most once*, *at least once*, and *exactly once* services [17]. Reliability may also be increased by replicating the services on multiple hosts to make them more readily available, even if a host is unavailable for either internal or external reasons.

- *Scalability* — In this context, scalability is the ability to manage a growing load in the future. In fact, in centralized systems the load is limited by the load the node can support, but in distributed systems a new load request can be directed toward different servers according to necessity, while the user must be unaware of the real used resource. The typical mechanism used to accomplish this issue is *transparency* (i.e., location, migration, access, and replication transparency) [17,34]. Particularly important is *location transparency*, which demands that components do not know the physical location of the components with which they interact. A detailed discussion on transparency can be found in Emmerich [17]. For efficient location transparency, a load-balancing mechanism must be provided to either reduce or increase load on different hosts when a service is started or stopped somewhere in the distributed system.
- *Security* — Because the Internet is an important component of the actual communication network, it is not possible to protect connections from third parties. Sometimes, the data involved in distributed computations is private data, and users want to protect it from unauthorized access. The middleware itself should be able to provide cryptography mechanisms to the users.

Another issue is verifying the real identity of users to protect the server from unauthorized people and to ensure the high quality of a given service for the client. At least four security services can be incorporated:

- Protection of data against reading by unauthorized users
- Forbidding the creation and deletion of messages to unauthorized users
- Identity checking
- Possibility of using electronic signed documents

How Mobility Affects Middleware Design

The rapid growth of wireless technologies and the development of smaller and smaller devices have led to the widespread use of mobile computing. Each user, equipped with a portable device, is able to access miscellaneous

services in any way at any time and anywhere thanks to the connectivity powered by modern network technologies. Mobile access to distributed applications and services raises many new issues. A major problem is the wireless technology itself, as the bandwidth provided is orders of magnitude lower than in the wired networks, signal loss is very frequent, and the noise level is influenced by external conditions. A second aspect is related to the mobile devices, which are characterized by scarce resources in terms of CPU, RAM, display, and storage; in particular, they are equipped with smart batteries, which limit the autonomy of the device (in terms of power consumption) and affect both wireless transmission and access to services that require a high computational load. Finally, a third aspect that must be considered is user mobility, which causes problems related to signal loss during movement to a new cell (handoff), as well as problems with address management caused by users traversing through different administrative domains and the need to adapt services to the position of the user.

Traditional middleware solutions are not able to adequately manage these issues. Originally designed for use in a static context, such middleware systems hide low-level network details to provide a high level of transparency to applications. In mobile environments, though, the context is extremely dynamic and cannot be managed by *a priori* assumptions, so it is necessary to implement reconfiguration techniques that can react to changes in the operating context and develop powerful mechanisms to propagate such changes until the application level is reached. It is necessary, therefore, to develop new middleware for mobile systems in the early stages of the design phase that will provide mobility support.

Mobile Middleware Requirements

To highlight the issues related to the mobile middleware design we present a typical mobile computing scenario, focusing on the main aspects that must be taken into account. Consider the case of a user equipped with a personal digital assistant (PDA) who wishes to receive information about movies playing at the nearest cinema. The user will select a movie after viewing some video clips. To meet the user's requirements, the system must:

- Determine the user's actual position, search for the requested services, and select the ones more appropriate for the user.
- Be aware of the user's habits (e.g., the user's favorite movie genre), so it can select the results to be presented to the user.
- Establish and rate the quality of service (QoS) level to be used.
- Format the information to be presented according to the hardware and software features of the device used.

- Manage the user's mobility by allocating the resources in advance for using the service (e.g., the bandwidth required for streaming) and managing the changes of address.
- Manage the load of the wired network so as to optimize the resources by changing the server from which the streaming will be retrieved.

It may happen that the user goes from a WiFi area to a General Packet Radio Service (GPRS) area with a consequent decrease in the bandwidth available, or the user might change devices (e.g., the user may switch to a PC upon arriving home). In these cases, the system should be able to reconfigure the parameters to resume the view from the point where it was stopped and to adapt it to the new device. Some operations of clip transcoding, reallocation of resources, and QoS management will therefore be necessary. When the movie has been chosen, the user may decide to buy tickets online, in which case the system should be able to manage the transaction with regard to issues of both security and the possible disconnection and reconnection of the user.

Using our example of the moviegoer, some interesting issues involved in the management of advanced services in mobile environments can be identified, such as service discovery, QoS, service adaptation, and load balancing. Service discovery plays a primary role, because it allows the user to access the list of available services. The techniques currently used include the use of distributed directory services, which are more scalable than the centralized approach [30]. The user position and the user profile will have to be managed to filter the services available by choosing, initially, only those corresponding to the user preferences and habits and located near the user's actual position.

The streaming of multimedia flows over wireless channels requires very strict QoS requirements. In fact, any variation in the quality of the transmission can degrade the application requested and make it inaccessible; for example, when a user is viewing a movie clip, the bit rate cannot be reduced below a specific threshold without affecting the viewing quality. The QoS management is therefore very important.

A major problem is the limited bandwidth available in wireless channels. This involves the investigation of new techniques for bandwidth reservation to anticipate the user's handoff and reallocate resources in the new access point that will be visited by the user [11]. The system cannot disregard the user's actual position, as it has to propose services and resources available near the user's position. This is necessary because of restrictions in terms of QoS and because a crucial factor is the distance (latency) between the server and the mobile client.

Furthermore, the system will have to be able to recognize features of the device to adapt the service requested to the type of terminal used; for example, if the same movie clip is to be viewed on a notebook with a 16.8-million-color display and on a PDA with a 256-color display, different resolutions will be required. Considering the bandwidth savings required in wireless environments, the need to transcode the same video in several formats to tailor it according to the client device is evident; for example, an MPEG-2 video could be encoded in MPEG-4 when switching from a wired to a wireless station. Information can be distributed on the wired network, which allows quicker and more effective access to the resources and makes available more copies of the same video, even in different formats.

Load-balancing operations are necessary to better exploit the use of storage and computing resources. Wireless channels are characterized by frequent disconnections, which cause QoS degradation and possible failures due to data loss during a transaction; therefore, the system should be able to manage such sudden disconnections by providing mechanisms for information replication and transaction recovery. Finally, all of these operations must be carried out taking into consideration the limited resources of mobile devices, particularly power consumption. Some power-saving techniques must be applied to reduce the wireless transmission load during both downloads and uploads [3].

In light of the many issues that must be addressed when developing mobile middleware, it can be observed that the use of traditional middleware originally conceived for fixed distributed systems is not always feasible in such complex and heterogeneous environments. This is due primarily to the remarkable differences between the two operating environments (see Table 7.1). These differences call for new design strategies that take into account features of the mobile environment to overcome,

Table 7.1 Main Differences Between Distributed and Mobile Environments

	Distributed Environments	*Mobile Environments*
Bandwidth	High	Low
Context	Static	Dynamic
Connection type	Stable	Unstable
Mobility	No	Yes
Communication	Synchronous	Asynchronous
Resource availability	High	Low

in an efficient way, all of the issues discussed here [16,31]. Such design strategies can be classified into three main types: context management, connection management, and resource management.

Context Management

The main assumption of middleware for fixed distributed systems (i.e., static representation of the context that is not transparent to upper layers) is too restrictive in mobile environments, as such environments are characterized by frequent context changes. Mobile computing applications must adapt their behavior to these changes to overcome issues related to high-level service provisioning. Context transparency, in fact, makes the development of complex applications easier, but it does not allow the service level to make decisions about the environments in which such applications must run [14]. This approach is powerful in systems where the operating conditions are static or where changes can be considered as exceptional and predictable events.

Mobile environments do not satisfy these requirements. Disconnections can occur either voluntarily (due to power-saving policies) or suddenly (due to signal loss), wireless technologies differ greatly in terms of performances and reliability, and portable devices have a high degree of heterogeneity. To enable applications to adapt to such context evolutions, parameter reconfiguration at provision time is necessary [4,13]. Mobile middleware systems cannot hide the context at the upper layers but instead must both represent it and announce changes until the service layer is reached. It is at this layer, in fact, that is possible to make decisions about the best way to react to context changes. For example, in a multimedia streaming session, in response to a drastic reduction in bandwidth, the system could activate the movie transcoding tasks (by reducing the bit rate or the color depth) or could decide to transmit only the audio data, dropping all the video packets (possible cases might include football matches, video conferences, video clips). Clearly, such decisions can be made only when the service typology is known and not on the basis of low-level information. It is necessary to implement middleware systems in such a way as to achieve a trade-off between transparency and awareness [12].

The operating context in a mobile computing scenario can be divided into three main aspects: user context, device context, and network context. The user context is composed of information related to the user's position, to user preferences, and to the QoS level requested. The device context includes details on the status of the available resources (e.g., CPU, batteries, display) and on the relative position of the device in the network — for example, in terms of latency between the device and a service

Table 7.2 Context Representation

Context Type	*Description*
User context:	
Location	Real position of the user in the wireless environment
Profile	User preferences and habits
QoS level	QoS level requested by the user accessing the services
Device context:	
Profile	Device features (e.g., CPU type, display)
Resource status	Updated usage level of the device resource
Location	Latency from other devices and from the service providers
Network context:	
Technology	Adopted wireless technology (e.g., 802.11, Bluetooth, GPRS)
Noise level	Quality level of the wireless signal
Activity	Wireless activity in terms of throughput
Bandwidth available	Bandwidth available in the wireless environment
Addressing	Address protocol used by the wireless infrastructure

provider or in terms of distance from other hardware components (e.g., mobile devices, printers). The network context contains all the information about the available bandwidth, noise level, wireless technologies adopted (e.g., 802.11, Bluetooth®), and addressing. Such context representation is shown in Table 7.2.

One of the main design considerations of context-aware middleware is the study of a representation of the operating context to capture its features and to make them available to the upper layers. Such a representation has to be flexible and powerful to allow applications to easily react at provision time to the frequent context changes. A common technique adopted for context representation is the definition of metadata; in particular, profiles and policies comprise metadata, which describes with a high level of abstraction context features and actions to carry out in case of changes. The metadata, represented by a meta-language (e.g., XML [1]), has to be separated from the implementation details to simplify the management operations. Suitable binding techniques must be impslemented to enable applications to change their execution at runtime according to the established policies. Such requirements have made computational reflection, introduced in Smith [39], an attractive technique to adopt in the mobile

middleware design. Reflection is the ability of a software system to monitor its computation and to change, if necessary, the way it is executed. The two phases of monitoring and adapting are generally referred to as *introspection* and *interception*. Discussions of context-aware middleware based on the concept of reflection can be found in the literature [2,29].

A context feature that has aroused quite a bit of interest in recent years is location management [25]. Location-aware middleware that is able to provide services according to the user's position and to manage the user's movements has been developed for such scenarios as e-health [37], e-learning [24], and cultural heritage [28].

Connection Management

User mobility, intermittent signals, and resource management policies give rise to the frequent disconnection and reconnection of mobile devices. Such behavior, which is not encountered in traditional distributed systems, makes unsuitable the adoption of a synchronous communication system that is based on the assumption that the sender and receiver are continuously connected during the communication phases. Mobile middleware has to provide an asynchronous communication system able to carry out all the tasks, notwithstanding the intermittent link between sender and receiver. To this end, solutions to decouple the sender and the receiver are required. Decoupled middleware systems have to manage the issues related to data synchronization by implementing data replication techniques. One of the most widely adopted solutions is the use of tuple space systems, which provide shared memory areas where both the sender and the receiver can put their data in an asynchronous way [33]. When a message has been sent (that is, after a *write* operation), the sender can continue its tasks without waiting for the receiver to carry out the *read* operation; thus, a mobile user can make a query, disconnect from the network, and, when reconnected, retrieve the results of that query. Examples of tuple-space-based middleware include TSpaces™ [43] and JavaSpaces™ [22].

Another technique adopted for transaction management in mobile environments is the use of data subsets downloaded in the mobile device to create a local representation of information scattered over the wired network; offline transactions can be carried out by mobile users using these local data subsets, and the actual operations can be carried out when the user goes online [18,38]. For example, these subsets can be adopted in an e-commerce scenario to allow users to download a part of the product list and to create, offline, a local shopping cart with the selected products. When an online connection is established, the system will retrieve the local shopping cart to complete the order.

A drawback of such solutions is the data synchronization that requires the use of advanced techniques; in our offline shopping scenario, we have to deal, for example, with issues related to potential price updates on the official product list that are not indicated on the older, local product list, or we might have to take into account problems related to product availability. The system should be able to disseminate these updates and to carry out effective comparisons among data, verifying the correctness of the transactions. Such operations depend greatly on the manner in which the data is structured.

Another issue related to connection management is the provision of services based on the concept of session (e.g., multimedia streaming). A temporary disconnection or change of address could cause the loss of the session and the end of service provisioning. Such issues can be solved by adopting proxies capable of decoupling the client and server and hiding these disconnections from the service layer [5,9]. Proxies have to interact with the specific protocol involved in service provisioning, and then their development is strictly related to the particular typology of the service that we want to use.

Resource Management

The design of mobile middleware and the management of the discussed techniques are strictly related to the hardware resources to be used for their execution, which are usually quite scarce, thus introducing another constraint on the design of such systems: Mobile middleware has to be lightweight [44]. Mobile middleware, on the one hand, has to implement techniques capable of guaranteeing powerful usage of available resources by reducing, for example, wireless transmissions and by adapting service typology to the real features of the client devices. On the other hand, mobile middleware has to be designed to be efficient to avoid overloading the device itself.

First, we must take into account the use of the sensors required to accomplish the goals of the middleware; an appropriate context representation, in fact, foresees the use of several sensors (e.g., of position) for the collection of the data that must be monitored to manage the context evolutions. The use of sensors must to be restricted as much as possible because such components are quite greedy in terms of resource consumption; for example, if location management is needed, triangulation techniques could be implemented rather than using global position system (GPS) modules on the mobile devices [26].

A second aspect is related to the computational load of middleware; in addition to being light in terms of memory usage, mobile middleware

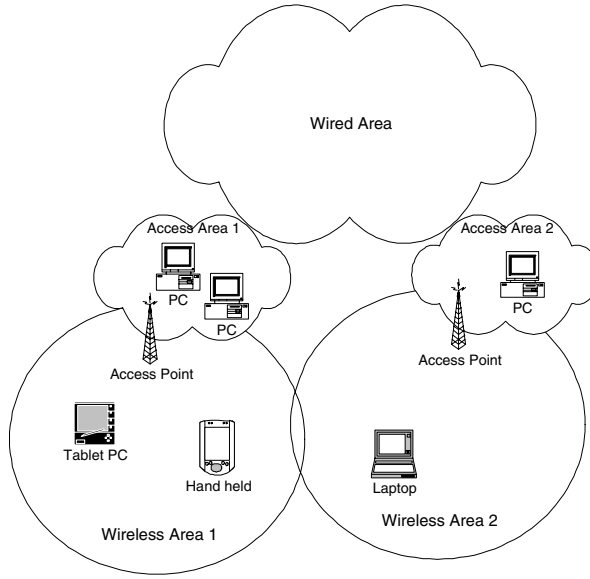

Figure 7.6 A nomadic computing scenario.

must reduce the amount of data to process so the limited computing resources of mobile devices are not overloaded. To this end, it is important to design highly modular middleware systems that are capable of activating only the modules absolutely necessary for the required operations (deactivating at runtime the unnecessary ones) [4] and delegating to other parts of the system (e.g., the wired infrastructure) those operations that require a high computational load, such as multimedia content tailoring [9].

Middleware for Nomadic Systems

Nomadic systems are characterized by a fixed infrastructure that provides a wireless access to mobile devices through access points. Thanks to this wireless link, mobile users are able to access services offered in the wired network. Designers of mobile middleware for nomadic systems must take into consideration many issues that involve both the services offered (in terms of their heterogeneity and complexity) and the main features (performance and dependability) of the processing and storage equipments. A typical nomadic computing scenario is shown in Figure 7.6. Such a setup includes the following areas:

- Wireless
- Access
- Wired

The wireless area is the coverage area of an access point, where one or more mobile devices can be found. The access area is the contact area between the wireless area and the wired area. It consists of access points that allow mobile devices to access services available in the wired area. The wired area is the core network infrastructure.

Managing the issues related to service provisioning in wireless environments calls for a middleware solution that covers all the areas presented in this scenario [6]. In such a way, interactions between the wired and the wireless components of the system can occur. Moreover, the distribution of this middleware over the three described areas will allow carrying out expensive tasks whenever more convenient resources are available. By responding to the resource management issues, mobile middleware will never overload mobile devices.

It is in the wired area where all the tasks requiring a high computational load must be carried out. Such tasks are related to service adaptation, data storage, and load balancing. Powerful management of these tasks is mandatory to make effective the QoS strategies provided by the other areas of the scenario and to improve service provisioning [45]. Let us consider, for example, tailoring a multimedia format to the features of a client device. It can be easily observed that when we send high-resolution multimedia data to a smart device we will experience a high percentage of wasted resources, affecting other users present in the same cell and making useless any resource reservation policy.

The access area contains tasks related to resource reservation and context management. Resource reservation can be performed by admission-control techniques that restrict access to the wireless environment. To guarantee service provisioning during user movement, the middleware has to be able to reserve in advance resources in the new access point where the user is headed [10]. The advanced reservation strategies must know the user position and manage the bandwidth of each cell by leaving a portion of it for users coming from neighboring cells. QoS levels have to be created to allow bandwidth reconfiguration according to the users' needs and resource availability. The middleware, at provision time, can automatically adapt the bandwidth assigned to a user to accept new users from neighboring cells, thus reducing the call-blocking probability. Proxy solutions able to carry out tasks on behalf of the user have to be inserted in this area to overcome issues related to signal loss [5,11].

The wireless area hosts middleware components related to mobile devices. Due to the limited resources of such devices, these middleware

components must be reduced to only an interface with the system. In this area, we can find mobility management modules (only in the case of GPS systems), user and device profile management modules, the graphical interface for user–device interactions, and a communication mechanism that exchanges data with other middleware components present in the scenario.

Middleware for Ad Hoc Systems

Ad hoc systems are communication infrastructures where users can communicate notwithstanding previous agreements and can connect, disconnect, or move around in the surrounding space. This means that the nodes that form the network cannot rely on a fixed infrastructure nor on a central coordinating entity, because they all have the same computational potential and the same probability of disconnecting or migrating. To ensure the exchange of information among users, some routing protocols must be created. Such protocols route the packets in multi-hop paths, which consider a changing network topology.

The development of a mobile middleware for *ad hoc* systems is heavily influenced by the features of such environments. In fact, the lack of a fixed infrastructure limits most design choices. Any type of centralization has to be removed, as the presence of a static entity capable of carrying out tasks such as discovery of service, QoS management, and so on is not allowed. All of these tasks have to be accomplished using distributed approaches. Also, the mobile middleware implementation has to take into account the high degree of mobility typical of such environments and must produce an exhaustive context representation. The lack of a wired infrastructure restricts the execution of such middleware to mobile devices, as any distribution over different areas of the scenario, such as in the case of nomadic systems, is not possible. For this reason, the design of middleware for *ad hoc* systems has the main goal of achieving a high level of simplicity [44]. The most resource-expensive tasks cannot be delegated to other network components scattered in the system; instead, it is necessary to implement techniques that can accomplish the middleware operations using only the limited resources provided by mobile devices.

Although the development of mobile middleware for *ad hoc* systems is still in the early stages, some solutions and some guidelines are presented in the literature. Some authors have tried to modify middleware for nomadic systems to operate in such complex environments [20]; others have designed entirely new paradigms adapted to the features of *ad hoc* networks [35]. Peer-to-peer (P2P) is one of the most used paradigms; it is based on the concept of information dissemination between nodes and

on the use of advanced techniques searching and provisioning services in accordance with the network topology [27].

To overcome the resource consumption issues, cooperation techniques between nodes have to be developed such that it will be possible to utilize the least-frequently used devices for accomplishing complex tasks. These operations can be provided by designing middleware systems with a high degree of modularity which are able to manage loading and unloading operations at runtime and execute the component middleware in a parallel way.

Available Technologies for Mobile Middleware

Mobile middleware design calls for new network technologies able to manage the continuous changes of the environment and to provide cooperative mechanisms that overcome the lack of resources of portable devices. In this section, we show how the use of the mobile agent technology and of the grid computing paradigm provides an effective strategy in the deployment of an overall architecture that achieves the goals of middleware.

Mobile Agent Technology

A software agent is a kind of software package that is smart enough to act as an assistant to accomplish some tasks on behalf of human beings [19]. The most salient feature that distinguishes agents and ordinary code is autonomy. Agents can cooperate with other agents to carry out more complex tasks than they themselves could handle. One special kind of agent, the mobile agent, can move from one system to another to access remote resources or even to meet other agents. The big success of mobile agents can be attributed to their ability to combine the typical features of software agents (e.g., autonomy, delegation) with the opportunity to migrate by moving from one position to another [42]. On the one hand, this feature allows the operations to be decentralized; on the other, interaction is possible with the environment around the agent. The scalability of the system can be increased, and some context- and location-aware mechanisms can be used.

Mobile agents seem to be a natural choice for dealing with issues related to providing advanced services in mobile environments. Agent programming technologies have emerged as a flexible and complementary way to manage the resources of distributed systems due to the increased flexibility in adapting to the dynamically changing requirements of such systems [15]. Such technology is considered to be both promising and challenging with

regard to addressing personal or terminal mobility issues. Mobile agents are considered to be an enabling technology for automated, flexible, and customized service provisioning in a highly distributed way, as network nodes become active and take part in the computation of applications and provisioning of customized services. In addition to the clear separation of key functionality and aspects of deployment on the functional side, such technology offers potential technical advantages. Among them are reduced communication cost, reduced bandwidth usage, the possibility of using remote interfaces, and support for offline computation.

Mobile agents enable both temporal (i.e., over time) and spatial (i.e., over different nodes of the network) distributions of the service logic. These distributions add another technical advantage (namely, scalability), while at the same time such bottlenecks of centralized approaches as reduced network availability and malfunctioning are avoided. What makes this approach so appealing is how the previously discussed benefits of mobile agents address the typical issues and restrictions of wireless communication (e.g., low-bandwidth, high-latency networks; high bit error rate; low processing power; small area available for the user interface).

Grid Computing Paradigm

When developing a powerful infrastructure that must provide services of a guaranteed quality while maintaining the user's profile and addressing characteristics of mobile devices and the limited availability of resources, it becomes apparent that the wired and the wireless components are not as different as they might appear to be at first. In fact, strong coordination between these two environments is necessary. The grid computing paradigm [21] is a valid solution to implement distributed management strategies in the wired part of the system, which must strongly interact with the mechanisms available in the wireless part to provide ever more sophisticated services with a high level of QoS. It is extremely important to develop an infrastructure that provides effective QoS management and allows mobile users to benefit from the service requested, regardless of the device used and the users' moves.

Grid refers to a new distributed computational infrastructure that provides an innovative method for accessing and distributing data and resources. The idea on which the grid is based is allowing people to share transparently and on a wide scale computational data and resources with members of communities working toward the same purposes. Interest in the grid technology has been growing, primarily among scientific communities. The grid technology arose from the wide use of Internet computing. The grid makes use of the idle resources available on the Internet to perform distributed computational operations.

The grid is intended to provide a more rational use of the resources distributed on the network. This rational use includes many operations such as load balancing, QoS management, and secure access. Incorporation of the grid in the design of mobile middleware systems would make possible such operations as:

- *More effective management of distributed data in the network* (e.g., databases, online libraries, video-clips) — This offers the opportunity of moving data to bring it closer to the user or to manage overload and fault tolerance situations.
- *Management of services according to the user's device* — For the same user to access services through different environments, the terminal equipment may have to be different; for example, a terminal with a high-resolution screen may be desirable at home, but a handheld terminal with a low-resolution screen may be the cost of mobility. Clearly, as the environment changes, the content to be delivered to the user also changes. A video conference, for example, may be delivered as video and voice communications at a fixed terminal or as voice and text only at a mobile terminal.
- *Service discovery* — It would be possible to offer this feature through the use of distributed strategies based on data replication.

Grid computing would seem to be a powerful strategy for the development of middleware for *ad hoc* environments. In fact, the use of cooperative and resource-sharing techniques between nodes is mandatory in environments not equipped with any wired infrastructure.

The so-called *wireless grid* [32] is a new research field aimed at introducing grid computing concepts to systems composed of resource-constrained devices to carry out complex tasks in a parallel way by sharing the resources of idle devices. The peer-to-peer paradigm is used to discover the services and the resources provided by the devices, and component-based programming techniques are adopted to bind at runtime the software modules shared by nodes. At the moment, the use of a mobile grid in the design of mobile middleware for *ad hoc* environments appears to be one of the most interesting directions to pursue.

References

[1] eXtensible Markup Language (XML), http://www.w3.org/XML/.
[2] OpenORB Project, http://openorb.sourceforge.net.
[3] Anastasi, G., Conti, M., Gregori, E., and Passarella, A., A performance study of power-saving policies for Wi-Fi hotspots, *Computer Networks*, 45, 295–318, 2004.

[4] Bellavista, P., Corradi, A., Montanari, R., and Stefanelli, C., Context-aware middleware for resource management in the wireless Internet, *IEEE Trans. Software Eng.*, 29(12), 1086–1099, 2003.
[5] Bellavista, P., Corradi, A., and Stefanelli, C., Mobile agent middleware for mobile computing, *IEEE Comput.*, 34(3), 73–81, 2001.
[6] Bellavista, P., Corradi, A., and Stefanelli, C., Application-level QoS control for video-on-demand, *IEEE Internet Comput.*, 7(6), 16–24, 2003.
[7] Birrell, A.D. and Nelson, B.J., Implementing remote procedure calls, *ACM Trans. Comput. Syst.*, 2, 39–59, 1984.
[8] Box, D., *Essential COM*, Addison-Wesley, Boston, MA, 1998.
[9] Bruneo, D., Villari, M., Zaia, A., and Puliafito, A., VoD services for mobile wireless devices, in *Proc. of the IEEE Symp. on Computers and Communications (ISCC'2003)*, Antalya, Turkey, June 30–July 3, 2003, pp. 602–207,
[10] Bruneo, D., Paladina, L., Paone, M., and Puliafito, A., Resource reservation in mobile wireless networks, in *Proc. of the IEEE Symp. on Computers and Communications (ISCC'2004)*, Alexandria, Egypt, June 28–July 1, 2004, pp. 460–465.
[11] Bruneo, D., Villari, M., Zaia, A., and Puliafito, A., QoS management for MPEG-4 flows in wireless environment, *Microprocessors Microsyst.*, 27(2), 85–92, 2003.
[12] Capra, L., Emmerich, W., and Mascolo, C., Middleware for mobile computing: awareness vs. transparency, in *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schloss Elmau, Germany, May, 2001, pp. 164–169.
[13] Capra, L., Emmerich, W., and Mascolo, C., CARISMA: Context-aware reflective middleware system for mobile applications, *IEEE Trans. Software Eng.*, 29(10), 929–945, 2003.
[14] Chan, A. and Chuang, S.-N., MobiPADS: a reflective middleware for context-aware mobile computing. *IEEE Trans. Software Eng.*, 29(12), 1072–1085, 2003.
[15] La Corte, A., Puliafito, A., and Tomarchio, O., An agent-based framework for mobile users, in *Proc. of the European Research Seminar on Advances in Distributed Systems (ERSADS'99)*, Madeira, Portugal, April 23–28, 1999.
[16] Eliassen, F. et al., Next generation middleware: requirements, architecture and prototypes, in *Proc. of the 7th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'99)*, Capetown, South Africa, December 20–22, 1999, pp. 60–65, 1999.
[17] Emmerich, W., *Engineering Distributed Objects*, John Wiley & Sons, New York, 2000.
[18] Mascolo, C. et al., Xmiddle: a data-sharing middleware for mobile computing, *Wireless Pers. Commun. Int. J.*, 21(1), 77–103, 2002.
[19] Etzioni, O. and Weld, D.S., Intelligent agents on the Internet: fact, fiction, and forecast, *IEEE Expert*, 10(3), 44–49, 1995.
[20] Fok, C., Roman, G., and Hackmann, G., A lightweight coordination middleware for mobile computing, in *Proc. of the Sixth Int. Conf. on Coordination Models and Languages*, Pisa, Italy, February 24–27, 2004, pp. 135–151.
[21] Foster, I., Kesselman, C., and Tuecke, S., The anatomy of the grid: enabling scalable virtual organizations, *Int. J. Supercomputer Appl.*, 15(3), 200–222, 2001.

[22] Freeman, E., Hupfer, S., and Arnold, K., *JavaSpaces: Principles, Patterns, and Practice*, Addison-Wesley, Boston, MA, 1999.
[23] Gilman, L. and Schreiber, R., *Distributed Computing with IBM MQSeries*, John Wiley & Sons, New York, 1996.
[24] Griswold, W.G. et al., ActiveCampus: experiments in community-oriented ubiquitous computing, *IEEE Comput.*, 37, 73–81, 2004.
[25] Hazas, M., Scott, J., and Krumm, J., Location-aware computing comes of age, *IEEE Comput.*, 37, 95–97, 2004.
[26] Hightower, J. and Borriello, G., Location systems for ubiquitous computing, *IEEE Comput.*, 34(8), 57–66, 2001.
[27] Hsieh, H. and Sivakumar, R., On using peer-to-peer communication in cellular wireless data networks, *IEEE Trans. Mobile Comput.*, 3(1), 57–72, 2004.
[28] Krosche, J., Baldzer, J., and Boll, S., MobiDENK: mobile multimedia in monument conservation, *IEEE Multimedia*, 11, 72–77, 2004.
[29] Ledoux, T., OpenCorba: a reflective open brocker, in *Proc. of the Second Int. Conf. on Meta-Level Architectures and Reflection*, Vol. 1616, Lecture Notes in Computer Science, Springer, Berlin, 1999, pp. 197–214.
[30] Lee, D.L., Xu, J., Zheng, B., and Lee, W., Data management in location-dependent information services, *IEEE Pervasive Comput.*, 1(3), 65–72, 2002.
[31] Mascolo, C., Capra, L., and Emmerich, W., Middleware for mobile computing, in *Networking 2002 Tutorial Papers*, Vol. 2497, Lecture Notes on Computer Science, Springer, Berlin, 2002, pp. 20–58.
[32] McKnight, L.W., Howison, J., and Bradner, S., Wireless grids: distributed resource sharing by mobile, nomadic, and fixed devices, *IEEE Internet Comput.*, 8(4), 24–31, 2004.
[33] Nitzberg, B. and Lo, V., Distributed shared memory: a survey of issues and algorithms, *IEEE Comput.*, 24, 52–60, 1991.
[34] ISO, *Open Distributed Processing: Reference Model*, Technical Report ISO 10746-1, International Organization for Standardization, Geneva, Switzerland, 1998.
[35] Plagemann, T., Goebel, V., Griwodz, C., and Halvorsen, P., Towards middleware services for mobile *ad hoc* network applications, in *Proc. of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03)*, San Juan, Puerto Rico, May, 2003, pp. 249–255.
[36] Pope, A., *The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture*, Addison-Wesley, Boston, MA, 1998.
[37] Rodriguez, M.D., Favela, J., Martinez, E.A., and Munoz, M.A., Location-aware access to hospital information and services, *IEEE Trans. Inform. Technol. Biomed.*, 8(4), 448–455, 2004.
[38] Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., and Steere, D.C., CODA: a highly available file system for a distributed workstation environment, *IEEE Trans. Comput.*, 39(4), 447–459, 1990.
[39] Smith, B., Reflection and semantics in LISP, in *Proc. of the 11th Annual ACM Symp. on Principles of Programming Languages*, Salt Lake City, UT, January, 1984, pp. 23–35.

[40] *JavaSoft: Java Remote Method Invocation Specification*, Revision 1.5, jdk 1.2 edition, Sun Microsystems, Santa Clara, CA, 1992.
[41] Tanenbaum, A.S., *Computer Networks*, 4th ed., Prentice Hall, Upper Saddle River, NJ, 2002.
[42] Waldo, J., Mobile code, distributed computing, and agents, *IEEE Intelligent Syst.*, 16(2), 10–12, 2001.
[43] Wyckoff, P. et al., TSpaces, *IBM Syst. J.*, 37(3), 454–474, 1998.
[44] Yu, Y., Krishnamachari, B., and Prasanna, V.K., Issues in designing middleware for wireless sensor networks, *IEEE Network*, 18(1), 15–21, 2004.
[45] Zaia, A., Bruneo, D., and Puliafito, A., A scalable grid-based multimedia server, in *Proc. of Workshop on Emerging Technologies for Next Generation GRID (ETNGRID-2004)*, Modena, Italy, June 14–16, 2004, pp. 337–342.