UNIVERSITÀ DEGLI STUDI DI PARMA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Dottorato di Ricerca in Tecnologie dell'Informazione
XXII Ciclo

Maria Chiara Laghi

SERVICE ORIENTED MOBILE COMPUTING

DISSERTAZIONE PRESENTATA PER IL CONSEGUIMENTO
DEL TITOLO DI DOTTORE DI RICERCA

GENNAIO 2010

UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXII Ciclo

SERVICE ORIENTED MOBILE COMPUTING

Coordinatore:

Chiar.mo Prof. Carlo Morandi

Tutor:

Chiar.mo Prof. Gianni Conte

Dottorando: *Maria Chiara Laghi*

Gennaio 2010

To my family
and my friends

Contents

List of Figures

Introduction

Pervasive Computing (PC), also referred to as Ubiquitous Computing, is an emerging paradigm concerning with the increasing integration of ICT into people's lives and environments, made possible by the growing availability of microprocessors with inbuilt communication facilities. In other words, Pervasive Computing deals with the design of context-aware, adaptive applications, allowing also information access *anywhere, anytime from any device*. Pervasive computing has many potential applications, from health and home care, to environmental monitoring and intelligent transport systems.

Our research has focused on the integration of Mobile Computing and Service Oriented Architecture (SOA) in a Pervasive Computing perspective. We considered three interleaved concepts: the way people view mobile computing devices and use them within their environments to perform tasks, the way applications are created and deployed to enable such tasks to be performed, and how the environment is enhanced by the emergence and ubiquity of new information and functionalities.

Mobile Computing refers to a broad set of computing operations that allow a user to access information from portable devices such as laptop computers, PDAs, smart phones, handheld computer, music players, portable game devices, etc. Due to digital convergence, mobile industry is facing a significant disruption in these years. Multifunctional products are emerging for consumers, and diversification is introducing a new set of requirements for architectures and platforms, such as flexibility, scalability and modularity. Mobility is considered a strategic component of enterprise business, and deploying mobile applications provides great productivity improvements.

On the other hand, service discovery and composition are key concepts for developing pervasive applications. In a ubiquitous computing environment, a service-oriented infrastructure must be enabled with service discovery protocols (SDPs) to find the most appropriate services, either upon direct request from the users or proactively. In this context, we have considered the Web Service technology, that provides standard, simple and lightweight mechanisms for exchanging structured and typed information between services in a decentralized and distributed environment.

Therefore, to develop proactive and self-adaptive applications for ubiquitous and pervasive computing requires to merge information from diverse layers of a system to produce an effective response. To this purpose, it is necessary to identify the most suitable interaction paradigm. It is worth noting that most of development work effort to date is based on the traditional Client-Server interaction paradigm (CSP). However, this paradigm could be inadequate for ubiquitous and pervasive applications in particular when context- awareness, auto-adaptive and emergence functionalities are required. To improve scalability, we support the shift from traditional client/server architectures to systems based on the peer-to-peer (P2P) paradigm, completed by the self-organization and the self-adaptation principles.

The peer-to-peer paradigm enables two or more entities to collaborate spontaneously in a network of equals (peers) by using appropriate information and communication systems without the necessity for central coordination. Furthermore, a peer-to-peer system is a complex system, because it is composed of several interconnected parts that as a whole exhibit one or more properties (*i.e.* behavior) which cannot be easily inferred from the properties of the individual parts. In the context of education this trend may enable each user to be much more proactive in the creation of paths for his/her own training and in communicating with other members which share some interests to build new group knowledge.

Figure 1 illustrates how these paradigms fit together. SOA separates functions into distinct units (services), which can be distributed over a network and can be combined and reused to create applications. Peer-to-peer allows flexible and scalable resource sharing, including services. Mobile computing addresses the mobility of user, context-awareness and interaction with the environment. The state of the art in

these research fields is illustrated in section 1.

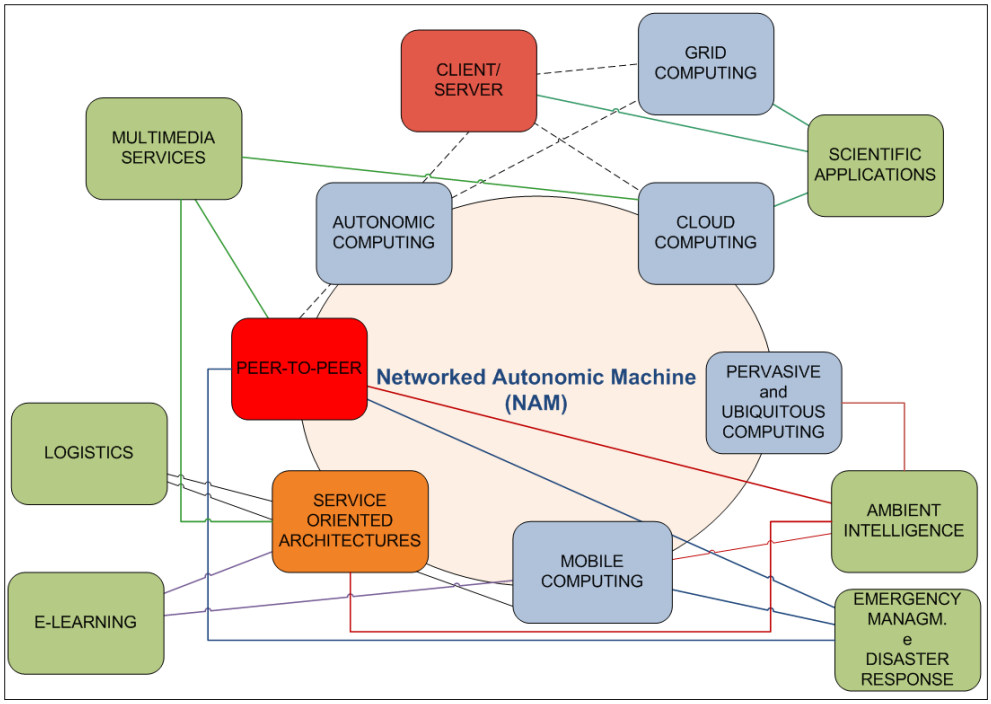

Figure 1: Convergence of application contexts and emerging paradigms within NAM framework

In order to enforce SOA and peer-to-peer functionalities to devices with limited processing and storage resources, a lightweight, modular middleware is required. While developing such a tool (illustrated in section 3), we have defined a novel formalism to describe the envisioned pervasive systems. Indeed, the Networked Autonomic Machine (NAM) is a theoretical model of an hardware/software entity that is programmed to be altruistic in sharing its resources. In particular, the focus is on special kinds of resources, *i.e.* services, offered to and by mobile devices. In section 2 we present NAM and its service-oriented specialization, called NSAM, a framework for peer-to-peer service sharing, based on three key aspects: overlay scheme, dynamic service composition and self-configuration of peers.

Interestingly, we have found that NAM and NSAM are suitable not only for Pervasive Computing, but also for another emerging paradigm, *i.e.* Cloud Computing (CC), in which dynamically scalable and often virtualized resources are provided as a service over the Internet. CC mixes aspects of Grid computing, Internet computing, Utility computing, Autonomic computing, Edge computing, Green computing and is strictly related to the SOA paradigm.

Scalable, self-managing and autonomic PC and CC systems should be characterized by

- Dynamic instantiation, dynamic composition, dynamic configuration, dynamic reconfiguration of services
- Vertically scaling platforms
- Self-configuration and adaptation of Cloud Computing platforms in response to anomalies of the run-time environment
- Self-managing platforms
- Horizontally scaling platforms

The resource mobility approach embedded in the NAM formal specification (which means *service code on-demand*, in NSAM) allows to design open, efficient and innovative cloud computing systems, with service providers forming a peer-to-peer network, and services being dynamically deployed and un-deployed, depending on the number of concurrent requests.

Chapter 1

State of the Art

Change is the only thing in
the world that is unchanging

– Heraclitus of Ephesus

The emergence of compact albeit powerful devices is giving users the ability to access, anytime and anywhere, globally available applications. While the anytime anywhere goal of mobile computing can be considered a reactive approach to the need for information access, pervasive computing is perceived as a proactive approach with the vision of "all the time everywhere". In 1991, Mark Weiser [1] described his vision of pervasive computing; the essence of that vision was the creation of environments saturated with computing and communication capability, yet gracefully integrated with human users thus becoming a ”technology that disappears.” Pervasive computing promises to build software, devices and networks that are deeply embedded in the user’s physical environment. In fact users may not even notice that they are dealing with a computing environment because the facilities are seamlessly and naturally integrated with the physical environment. Since motion is an integral part of everyday life, such technologies must support mobility; otherwise, a user will be aware of the technology by its absence when he/she moves. Augmented by the ubiquitous, heterogeneous and always-on communications and intelligent sensing of both devices and

environment, PDA and smart phones, as universal mobile terminals, are becoming integral part of the physical environment.

As stated in [2], pervasive computing represents a major evolutionary step in a line of work dating back to the mid-1970's. Two distinct related steps in this evolution are distributed systems and mobile computing.

The field of distributed systems arose at the intersection of personal computers and local area networks. A distributed system consists of a collection of autonomous computers, connected through a network and distribution middleware, which enables computers to coordinate their activities and to share the resources of the system, so that users perceive it as a single, integrated and coherent system. Distributed systems are mainly characterized by:

- *Openness*: the possibility to extend and modify the system easily, to respond to changed functional requirements. Any real distributed system will evolve during its lifetime.
- *Heterogeneity*: it calls for integration of components written using different programming languages, running on different operating systems, executing on different hardware platforms.
- *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.
- *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 , 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.

The appearance of full-function laptop computers and wireless LANs in the early 1990s led researchers to face the problems that arise in building a distributed system with mobile clients. Although many basic principles of distributed system design continued to apply, key constraints of mobility forced the development of specialized techniques. These constrain deals with the concepts of device, of network connection and of execution context. Devices in a fixed distributed system are stationary or fixed and vary from home PCs, to workstations, to mainframes, while a mobile distributed system has at least some physically mobile devices that range from personal digital assistants, to mobile phones,sensors, 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. Fixed hosts are usually 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 behavior of the system. The performance of wireless networks (*i.e.*, GSM, GPRS, UMTS and HSDPA networks, satellite links, WaveLAN, HiperLAN, Bluetooth) may vary depending on the protocols and technologies being used. 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. 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. 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. Because of these challenges, mobile computing is a very active and evolving field of research. An important issue is to determine the role of the mobile hosts in a distributed system. At one extreme, mobile hosts are used as dumb terminals and at the other they are used as workstations with enough computational power and memory to perform computa-

tions locally. In the former case mobile hosts will depend on some assigned host to compute on their behalf (assignment may change over time), while in the latter case, mobile hosts would be active participants of the distributed system. Mobility makes the location of the user a fast-changing parameter and the reliance on a fixed network configuration unrealistic. The ubiquity of mobile hosts results in computing systems that are of a much larger scale than the present distributed systems . This implies operating in a highly heterogeneous environment and raises questions of organizing information efficiently.

In determining the difference between software systems for mobile environments and those for static distributed environments, two different categories of problems are highlighted: the first refers to issues of portability and extensibility; on how existing systems and algorithms should be modified or augmented to be used in a mobile computing environment; the second refers to building and experimenting with new systems taking into account the fuzziness of the environment.

1.1 Pervasive and ubiquitous computing

Pervasive Computing (often considered the same as ubiquitous computing in the literature) is a related concept but can be distinguished in terms of interaction with the environment. The main objective of Ubiquitous Computing (UC) is to provide globally available services and resources in a network by giving users the ability to access them anytime and irrespective to their location. The main issue in Pervasive Computing (PC) is to provide spontaneous emergent services created on the fly by mobile or wireless devices that interact by ad hoc connections. In other words, Pervasive Computing deals with the design of adaptive applications that interact with the closest environment enhanced by context-awareness and emergence functionalities. Pervasive computing basically concerns three interleaved concepts: the way people view mobile computing devices, and use them within their environments to perform tasks, the way applications are created and deployed to enable such tasks to be performed and, finally, it concerns the environment and how it is enhanced by the emergence and ubiquity of new information and functionalities.

In [2] some research thrusts are related to pervasive computing. The first research thrust is the *use of smart spaces*. A space may be an enclosed area such as a meeting room or corridor, or it may be a well-defined open area such as a courtyard or a quadrangle. By embedding computing infrastructure in building infrastructure, a smart space brings together two worlds naturally disjoint. The second thrust is *invisibility*. The ideal expressed by Weiser is complete disappearance of pervasive computing technology from a user's conciousness. In practice, a reasonable approximation to this ideal is minimal user distraction. If a pervasive computing environment continuously meets user expectations and rarely presents him with surprises, it allows him to interact almost at a subconcious level. The third research thrust is *localized scalability*. As smart spaces grow in sophistication, the intensity of interactions between a user's personal computing space and his surroundings increases. This has severe bandwidth, energy and distraction implications for a wireless mobile user. The presence of multiple users will further complicate this problem. Scalability, in the broadest sense, is thus a critical problem in pervasive computing. Previous work on scalability has typically ignored physical distance, *i.e.* a web server or file server should handle as many clients as possible, regardless of whether they are located next door or across the country. The situation is very different in pervasive computing. The fourth thrust is the development of techniques for *masking uneven conditioning of environments*. The rate of penetration of pervasive computing technology into the infrastructure will vary considerably depending on many non-technical factors such as organizational structure, economics and business models.

The component technologies are very simple and basic. The hardware technologies (laptops, handhelds, wireless communication, software-controlled appliances, room cameras, and so on) are all here today. The component software technologies have also been demonstrated: location tracking, face recognition, speech recognition, online calendars, and so on. The real research issue is in the seamless integration of component technologies into a whole proactive system.

Proactivity is another central concept for pervasive computing; for proactivity to be effective, it is crucial that a pervasive computing system tracks user intent. Otherwise, it will be almost impossible to determine which system actions will help rather

than hinder the user. For example, suppose a user is viewing video over a network connection whose bandwidth suddenly drops. The system may reduce the fidelity of the video, pause briefly to find another higher-bandwidth connection, or advise the user that the task can no longer be accomplished. The correct choice will depend on what the user is trying to accomplish. Pervasive systems have to capture and exploit user intent and offer support for adaptation and proactivity. Adaptation is necessary when there is a significant mismatch between the supply and demand of a resource. The resource in question may be wireless network bandwidth, energy, computing cycles, memory, and so on. There are three alternative strategies for adaptation in pervasive computing. First, a client can guide applications in changing their behavior so that they use less of a scarce resource. This change usually reduces the user-perceived quality, or fidelity, of an application. Second, a client can ask the environment to guarantee a certain level of a resource. This is the approach typically used by reservation-based QoS systems. From the viewpoint of the client, this effectively increases the supply of a scarce resource to meet the client's demand. Third, a client can suggest a corrective action to the user. The existence of smart spaces suggests that some of the environments encountered by a user may be capable of accepting resource reservations. At the same time, uneven conditioning of environments suggests that a mobile client cannot rely solely on a reservation-based strategy when the environment is uncooperative or resource-impoverished, the client may have no choice but to ask applications to reduce their fidelities.

Another issue is relate on how powerful mobile client needs to be for a pervasive computing environment in terms of CPU, memory, disk capacity. Tipically a powerful client is named thick client while a thin client is a minimal one. Thick clients tend to be larger, heavier, require a bigger battery, and dissipate more heat, all negative factors from the viewpoint of the user who has to carry or wear the client. For a given application, the minimum acceptable thickness of a client is determined by the worst-case environmental conditions under which the application must run satisfactorily. A very thin client suffices if one can always count on high-bandwidth, low latency, wireless communication to nearby computing infrastructure, and if batteries can be recharged or replaced easily. If there exists even a single location visited by a

user where these assumptions do not hold, the client will have to be thick enough to compensate at that location.

A pervasive computing system that strives to be minimally intrusive has to be context-aware. In other words, it must be cognizant of its user's state and surroundings, and must modify its behavior based on this information. A user's context can be quite rich, consisting of attributes such as physical location, physiological state (such as body temperature and heart rate), emotional state (such as angry, distraught, or calm), personal history, daily behavioral patterns. If a human assistant was given such context, it would make decisions in a proactive fashion, anticipating user needs. More dynamic information has to be sensed in real time from the user's environment. Examples of such information include position, orientation, the identities of people nearby, locally observable objects and actions, and emotional and physiological state.

Finally, security issues have to be considered in a ubiquitous environment; privacy, in particular, already a thorny problem in distributed systems and mobile computing, is greatly complicated by pervasive computing. Mechanisms such as location tracking and smart spaces monitor user actions on an almost continuous basis. As a user becomes more dependent on a pervasive computing system, it becomes more knowledgeable about that user's movements, behavior patterns and habits. Exploiting this information is critical to successful proactivity and self-tuning. At the same time, unless use of this information is strictly controlled, it can be put to a variety of unsavory uses ranging from targeted spam to blackmail. Indeed, the potential for serious loss of privacy may deter knowledgeable users from using a pervasive computing system. Greater reliance on infrastructure means that a user must trust that infrastructure to a considerable extent. Conversely, the infrastructure needs to be confident of the user's identity and authorization level before responding to his requests. It is a difficult challenge to establish this mutual trust in a manner that is minimally intrusive and thus preserves invisibility.

1.1.1 Pervasive computing emerging paradigms

Therefore, to develop proactive and self-adaptive applications for ubiquitous and pervasive computing requires to merge information from diverse layers of a system to

produce an effective response. To this purpose, it is necessary to identify the most suitable interaction paradigm. It is worth noting that most of development work effort to date is based on the traditional Client-Server interaction paradigm (CSP). However, this paradigm could be inadequate for ubiquitous and pervasive applications in particular when context- awareness, auto-adaptive and emergence functionalities are required.

For challenging contexts such as ambient intelligence and emergency management, requiring highly efficient, pervasive and dependable solutions, a synergetic approach based on ubiquitous computing models and service-oriented technologies is envisioned. Moreover, to improve scalability, we support the shift from traditional client/server architectures to systems based on the peer-to-peer (P2P) paradigm, completed by the self-organization and the self-adaptation principles.

The peer-to-peer paradigm enables two or more entities to collaborate spontaneously in a network of equals (peers) by using appropriate information and communication systems without the necessity for central coordination. Furthermore, a peer-to-peer system is a complex system, because it is composed of several interconnected parts that as a whole exhibit one or more properties (*i.e.* behavior) which cannot be easily inferred from the properties of the individual parts.

At the beginning of the P2P era, Barkai (2002) proposed the following requirements for a general-purpose P2P middleware:

- portability item interoperability
- security
- local autonomy
- persistence
- scalability
- extensibility

With these objectives in mind, in recent years some researchers have focused on designing robust overlay schemes (with respect to bootstrapping, connectivity, mes-

sage routing) and distributed security / trust mechanisms, while others have targeted application-specific problems. Next step is to create decentralized and self-organizing infrastructures, being able to provide services to users according to their availability and the network status, and also supporting the spontaneous creation of services provided by heterogeneous nodes, such as mobile devices interacting through ad hoc connections without any prior planning. Gaber [3] has proposed two alternatives to the traditional client/server paradigm (CSP) to design and implement ubiquitous and pervasive applications: the Adaptive Services/Client Paradigm (SCP) and the Spontaneous Service Emergence Paradigm (SEP). In other words, the peer-to-peer paradigm is completed respectively by the self-organization and the self-adaptation principles. In SCP a decentralized and self-organizing middleware that implements an intelligent network should be able to provide services to users according to their availability and the network status. In SEP, spontaneous services can be created on the fly and be provided by mobile devices that interact through ad hoc connections without any prior planning. In order to enforce these paradigms to systems which include devices with limited processing and storage resources, lightweight middleware components are strongly required.

1.2 Mobile Computing

Mobile computing systems may be easily moved physically and their computing capabilities may be used while they are being moved. Examples are laptops, personal digital assistants, and mobile phones. Mobile computing systems differ from other computing systems in the tasks they are designed to perform, the way that they are designed, and the way in which they are operated. Among the distinguishing aspects of mobile computing systems, we recall their prevalent wireless network connectivity, their small size, the mobile nature of their use, their power sources, and their functionalities that are particularly suited to the mobile user. Because of these aspects, mobile computing applications are inherently different than applications written for use on stationary computing systems. Today it is difficult to imagine computing without network connectivity; networking and distributed computing are two of the largest

segments that are the focus of current efforts in computing. Wireless communication systems are often used in mobile computing systems to facilitate network connectivity, but they do not necessarily mean mobile, because some wireless technologies can only be used when one or both the parties are fixed or berely move during the wireless signal transmission, such as the short range bluetooth or infrared or long range wi-max. Wirelesse places an emphasys on the radio layer and link layer of the network protocol stack, whereas mobile computing takes advantage of ubiquitous wireless communications to build applications and services. The future of communication naturally involves the convergence of computing and communication in almost every aspects of information technology, thus allowing information access *anywhere, anytime from any device*. Not only information has to be easily accessible from any place and time, but is stored in a highly decentralized, distributed information infrastructure; a wide variety of information servers (both public and proprietary) needs to be accessible to mobile devices. Such devices themself may contain data, or data can be stored on flash memory smart-cards. This vision leads to the issue of building an universal mobile platform for reliable and high performance computing with heterogeneous seamless wireless access via limited computing resources.

Obviously any mobile computing system can also be stationary; therefore it is important to look at those elements that are outside of the stationary computing subset. In [4], the *dimensions of mobility* are defined as the tools that allow to qualify the problem of building software applications and mobile computing systems. Although these dimensions are not completely orthogonal with respect to each other, they are separate enough in nature to be distinguished and approximated to orthogonal variables. Moreover, some of these dimensions are limiting factors that usually are not considered when dealing with typical stationary application. These dimensions of mobility are:

- Location awareness
- Network connectivity quality of service
- Limited device capabilities (storage and CPU)

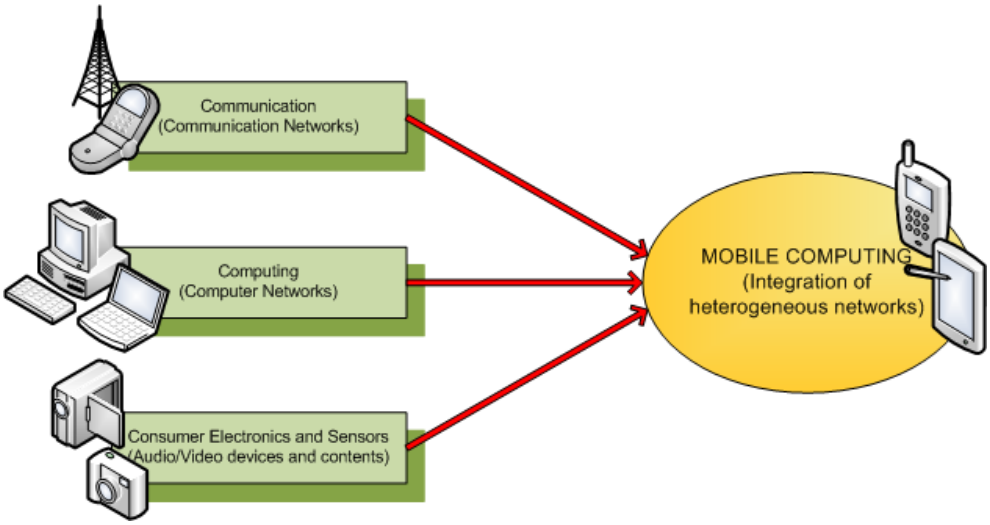

Figure 1.1: Digital convergence

- Limited power supply
- Support for a wide variety of user interfaces
- Platform proliferation
- Active transactions

A mobile device is not always at the same place, its location may be constantly changing. The challenges and opportunities presented by mobility can be divided in two categories: *localization* and *location sensitivity*. Localization is the mere ability of the architecture of the mobile application to introduce logic that allows the selection of different level of workflow, business logic and interfaces based on a given set of location information, commonly referred to as *locales*. Localization is not exclusive to mobile applications but takes a much more prominent role in mobile applications. Location sensitivity is something fairly exclusive to mobile applications. It is the ability of the device and the software application to first obtain location information while being used and then to take advantage of this location information

in offering features and functionality. It may also include the location of the device relative to some starting point or a fixed point, some history of past locations, and a variety of calculated values that may be found from the location and the time such as speed and acceleration. The most well known location sensing system today is GPS; GPS devices use triangulation techniques by triangulating data points from the satellite constellation that covers the entire surface of the earth. If a device does not have GPS capabilities but uses a cellular network for wireless connectivity, signal strength and triangulation or other methods can be used to come up with some approximate location information, depending on the cellular network. Actually, location information is one of the biggest drivers of mobile applications as it allows for the introduction of new business models and fundamentally new methods of adding productivity to business systems; location based mobile services can be applied in the field of emergency management, real-time traffic information updates (coupled with maps and navigation), mobile workforse monitoring (i.e logistic telematics), location based message services and computing service discovery and automatic configuration.

Whether *wired or wireless connectivity* is used, mobility means loss of network connectivity reliability. In the case of wireless network connectivity, physical conditions can significantly affect the *quality of service (QoS)*. QoS tools and products are typically used to quantify and qualify the reliability, or unreliability, of the connectivity to the network and are mostly used by network operators. Network operators control the physical layer of the network and provide the facilities, such as Internet Protocol (IP), for software application connectivity. Usually, the QoS tools, run by the network operators, provide information such as available bandwidth, risk of connectivity loss, and statistical measurements that allow software applications to make smart computing decisions. Stationary software applications typically assume some discrete modes of connectivity mostly limited to connected or disconnected. This works for most applications because most wired network connectivity is fairly reliable. Conversely, mobile applications have to know how to continue to operate even after they are disconnected from the network or while they connect and disconnect from the network intermittently and frequently. Almost all mobile applications should know how to stop working when the application suddenly disconnects from

the network and then resume working when it connects again. For example, the real-time bandwidth available should be part of the data provided and refreshed on some time interval. Such data can be used to design applications that dynamically adapt their features and functionality to the available bandwidth.

Another issue is related to the *dimension of devices*. Mobile devices are small and this physical size limitation imposes boundaries on volatile storage, non-volatile storage, and CPU on them. Smaller devices are easier to carry and, consequently, may become more pervasive. This pervasiveness also largely depends on the price of the devices. Making electronic devices very small normally increases the cost, as the research and development that go into making devices smaller are very expensive. But, once a technology matures and the manufacturing processes for making it becomes mostly automated, prices begin to decrease. At the point when the device is more and more of a commodity, smaller also means less expensive. Mobile applications must be designed to optimize the use of data storage and processing power of the device in terms of the application use by the user. The operating system of some devices may offer the available storage space, but this is not guaranteed. So, we need to design with the least amount of assumptions about the hardware capabilities of the device or with all those assumptions valid for all of the devices to be supported by the mobile application. A large part of engineering mobile applications requires first a theoretical understanding of the various types of platforms and operating systems available on mobile devices, then an understanding of the available commercial implementations of the varieties of types of operating systems and platforms and the type of applications best suited for each platform-device combination.

For the same set of reasons that wireless is the predominant method of network connectivity for mobile devices, batteries are the primary power source for mobile devices, thus introducing another constraint, namely a *limited power supply*; this must be balanced with the processing power, storage, and size constraints; the battery is typically the largest single source of weight in the mobile device. Mobile platforms allow the monitoring of the remaining power and other related power information. Some platforms allow multiprocessing and multithreading, which have an effect on the control over the variation of the CPU activity, which in turn has an effect on the

control over the power consumed by the device.

Mobile computing imposes another level of challenge on *human-computer interfaces (HCIs)*. Because applications and services are predominantly used on the move, the interaction between the user and the mobile terminal and between the user and a backend service system must be sufficiently simple, convenient, intuitive, flexible and efficient. In addition the user interface has to be designed to save power. Stationary users use non mobile applications while working on a PC or a similar device. The keyboard, mouse, and monitor have proved to be fairly efficient user interfaces for such applications. This is not at all true for mobile applications; examples of some alternative interfaces are voice user interfaces, smaller displays, stylus and other pointing devices, touch-screen displays, and miniature keyboards, that are often used in a combined way. Perhaps the biggest paradigm shift that designers and implementers of mobile applications must undergo is to understand the necessity of finding the best user interfaces for the application, architecting the system to accommodate the suitable user interfaces, implementing them, and keeping in mind that a new user interface may be required at any time. Key issues in the context of HCI are the following:

- *User interface design on a mobile device.* A number of interleaved factors as size, color depth, display resolution, battery time, weight, connection, etc., affect the design of a user interface for a mobile application.

- *Coherent user interfce design across multiple mobile devices.* Due to the lack of standardized representation layer of input and output devices, a user interface has to be designed and implemented on each type of targeting device. The layout of the GUI components and the operational logic must be consistent to allow easy navigation and intuitive operations.

- *Adaptive user interface design.* The application and supporting system software on a mobile device must be able to dynamically optimize the user interface on the device based on the hardware configuration of the mobile device and context of the running application.

Platform proliferation has very significant implications on the architecture, design, and development of mobile applications. Platform proliferation heightens the importance of designing and developing devices independent of the platform. Writing native code specific to the mobile device, unless absolutely necessary because of performance requirements, is not a recommended practice because of the proliferation of devices. The platform makers and manufacturers of devices and operating systems of those devices will always try to create restrictions on the developer to prohibit writing platform-independent applications. Regardless of the efforts of commercial platform builders, the software architects and developers should be focused on their primary task of meeting the user's requirements. And if these requirements include support of multiple platforms, which happens more frequently than not for mobile computing systems, platform independence should be on the top of the architects' and developers' list when choosing the tools to build an application.

Most of today's stationary applications have a restriction that can reduce the benefits of a mobile application system enormously: the user of the system must initiate all interactions with the system. We call such systems *passive systems* because they are in a passive state, waiting for some external signal from the user to tell them to start doing some particular thing. During the past two decades, messaging-based systems have been developed and have evolved. With messaging systems, each participant of the system can send a message to another participant and, if desired, under a specific topic in an asynchronous manner. Later came the idea of push; in the push model of communication, an information producer announces the availability of certain types of information, an interested consumer subscribes to this information, and the producer periodically publishes the information (pushes it to the consumer). Push systems, by definition, are *active systems*. For example, a particular user could be browsing the Web and, while purchasing some goods online, be notified of the change in the price of a particular stock. In this example, the system has taken an active role in starting communication with the user on a particular topic. *Active transactions* are those transactions initiated by the system; they may be synchronous or asynchronous. All active transactions are initiated by the system. Synchronous transactions are time-dependent transactions; the term is used to refer to a sequence of interactions between

the user and the computing system. Synchronous active transactions can be summarized by a set of properties:

- The transaction is initiated by the system, and during the same transaction, the user is given an opportunity, for a finite period of time, to respond to the action initiated by the system.
- Synchronous active transactions require a timely response from the user.
- The interactions between the system and the user work in a sequential and serial manner during a synchronous transaction.
- Synchronous active transactions are established between the system and a single user. This may be replicated for many users, but at the most elemental level, there is only one user in each active transaction.

Most of today's active systems are asynchronous. Asynchronous transactions are not time-dependent. Asynchronous active transactions, like their synchronous counterparts, can be described by a set of properties:

- Asynchronous active transactions work just like messaging systems. They can be established with either $1-n$ receivers or $1-n$ topics to which $1-m$ receivers are subscribed.
- Asynchronous active transactions may be a composition of $1-n$ messages sent by the system and may require $1-m$ messages back from the users. If $1-m$ messages required as responses from the users are not received within some time frame specified by the system, the transactions may be deemed as failed. Note that we are not defining the semantics of messaging systems (for if that is what we were referring to, we would be wrong). Rather, we are defining the semantics of asynchronous active transactions to be such that they encapsulate a number of messages being sent from the system to the user and from the user to the system and that some messages from the user, marked as responses to the messages from the system, can be required for the successful completion of the transaction.

Choosing whether the active behavior of a system is implemented using an asynchronous active transactional model or a synchronous active transactional model is completely dependent on the user requirements and the available tools, and is directly related to the available budget. Active transactions are an absolute essential part of mobile application development mainly because of the lack of focus on the part of the user while the user is mobile. It is also important to note that active transactions differ from push-pull systems and messaging systems not only because they can be both synchronous and asynchronous but also because they can contain $1-n$ interactions between the system and the user.

According with [5], the vision of mobile computing can be summarized in *convergence of mobile access* and *pervasiveness of mobile intelligence*(figure 1.2). The first addresses to the fact that system components and network elements, using a blend of wireless technologies, need to be coordinate and interoperable, prompting the use of mobile terminals that are able to provide applications and services with a set of sistematically integrated hardware and software components.

Moreover, because wireless is everywhere, access to backend systems and networks is ubiquitous; service and applications must be unobtrusive, meaning that the system of well-coordinated wireless components must be smart enough to choose the best way to accomodate a user's needs, and this process must be completely transparent to the user. The user does not need to use the system explicitly and the system helps the user in achieving a task automatically and intelligently. The trend of convergence of mobile access suggests unified access from a device to surrounding other wireless devices, whereas the pervasiveness of mobile intelligence represents how data are collected, processed, and disseminated among all the components in a mobile environment.

As explained in [4], the difference between mobile and stationary user can be referred to as *mobile condition*; the mobile user is fundamentally different from the stationary user in the following ways:

1. The mobile user is moving, at least occasionally, between known or unknown locations. The location of the user at a given time is a variable. Other variables may be the speed at which the mobile user may be traveling, what network con-

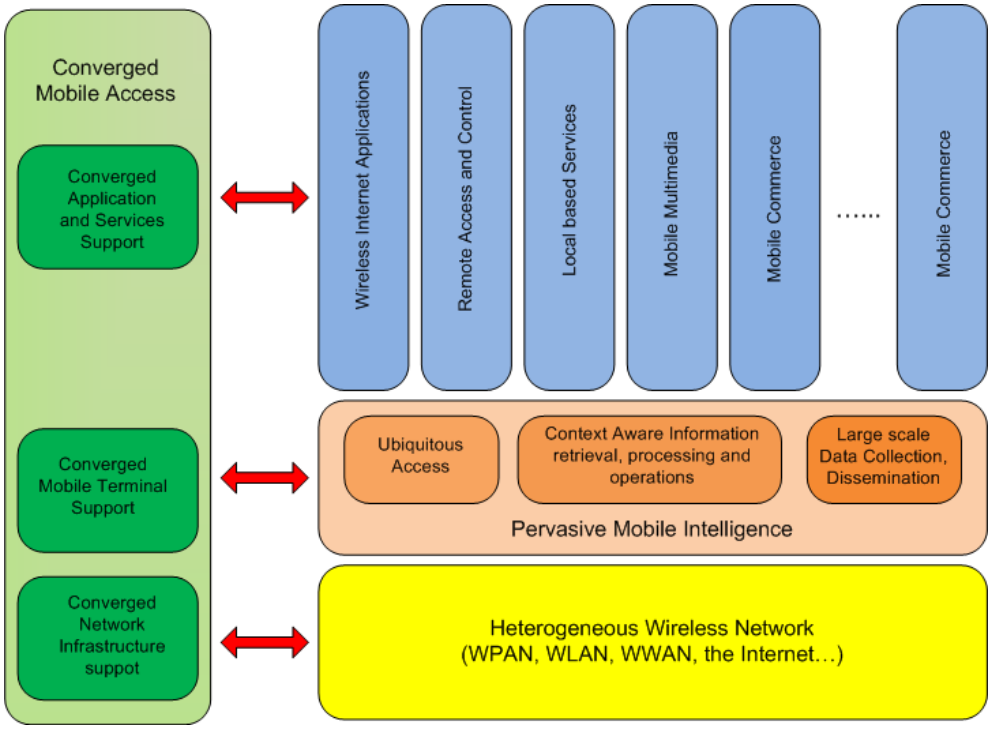

Figure 1.2: The vision of Mobile Computing

nectivity modes are available to the user, what the quality of that connectivity may be at any given place and time, or how long he or she may stay connected or disconnected. The mobile user also expects the system to have good connectivity coverage. The changing location of the mobile user also forces restrictions on power, size of device, wireless connectivity of the device, and just about every other aspect of the state of the mobile user.

2. The mobile user is typically not focused on the computing task. Mobile users are typically mobile because they are moving between two points with the primary task of reaching the destination. Another reason for lack of focus is multitasking; because of this multitasking nature of the mobile user, a variety of user interface input types such as voice may be needed to take advantage of

the senses that are not preoccupied by another task. Also, the user interface to the system must be very user friendly and require as few of the user's senses focused on communicating with the machine as efficiently as possible. For example, voice user interfaces allow users to focus on driving while still getting whatever information they need from the system.

3. The mobile user frequently requires high degrees of immediacy and responsiveness from the system. Mobile users are often in a situation where they need to quickly perform one or more computing tasks, such as retrieving contact information, sending a voice or e-mail message, or triggering some remote process. They don't have the time to go through a long boot sequence or long application setup times. Mobile users normally have higher expectations of performance from their devices than stationary users do. A short delay in application responsiveness can decrease its usefulness enormously.

4. The mobile user is changing tasks frequently and/or abruptly. The mobile user needs to be able to stop performing some computing task abruptly, do something that may be completely unrelated, then return to the application after some unknown period of time, and, without much effort to remember what he or she had been doing, continue the computing task. Mobile users expect applications that flow smoothly and do not require complex navigation despite the abrupt nature of their actions.

5. The mobile user expects to be able to retrieve data and do computing at any given moment and any given time. And this is precisely why the support for a variety of platforms with a variety of user interfaces is critical for a mobile application: to use an application anywhere and anytime, one may have to use it through whatever device is available and convenient for that given place and time. Mobile users expect to start a transaction and leave it unfinished on one device at a given place and time and finish the same transaction later on a different device and at a different place and time.

While designing mobile applications, it is important to balance the solutions to prob-

lems presented by each mobility dimension and to take into account the mobile condition for the mobile user. Once we the requirements from the user are defined, the first step in building the mobile application is to decide on the architecture; because access must be granted to the same application ubiquitously through any device and interface, mobile architectures are inherently network-based computing architectures that rely on suitable architectural patterns that take into account the described dimensions of mobility.

Due to digital convergence, mobile industry is facing a significant disruption in these years. Multifunctional products are emerging for consumers, and diversification is introducing a new set of requirements for architectures and platforms, such as flexibility, scalability and modularity. Mobility is considered a strategic component of enterprise business, and deploying mobile applications provides great productivity improvements. Mobility is complex, because it involves multiple back-end systems, some legacy, some newly deployed, and a collection of mobile devices with an increasing number of mobile operating systems (BlackBerry OS, Windows Mobile, Symbian OS, Mac OS X, Palm OS, Android and mobile Linux). A great variety of wireless technologies is also available in a global workplace, from current cellular networks with CDMA and GSM standards, to WiFi, WiMax, and future next-generation 4G networks.

1.2.1 Mobile hardware

As a small computer, a mobile device consists of integrated or interconnected hardware components and software. These hardware components include a microprocessor, read-only memory (ROM), random access memory (RAM), expansion storage, network interfaces and antenna, a battery, and a display. Some mobile devices also have a hard disk.

Mobile processors

A mobile device is controlled by a small, embedded, computer system; the notion of embedded system refers to a specialized computer system that performs a fixed set

of functions. Mobile devices such as cell phones, PDAs, laptop computers, pagers and wireless-enabled portable gaming devices constitute a large portion of embedded systems. The microprocessors (CPUs) used in mobile/portable devices are generally referred to as *mobile processors*, a type of embedded processor. The main characteristics of mobile processors are[6]:

- *Limited programmabiity* - a mobile processor can supply limited computing capability compared to a desktop computer, due to power cosumption constraints.
- *High I/O to computation ratio* - network and I/O communications are more frequent on mobile terminals than on desktop computers.
- *Stream and data processing* - multimedia processing is indispensable on many mobile terminals.

A few mobile processor families have emerged to become the major players in the mobile market.

The **Advanced Reduced Instruction Set Computer (RISC) Machine (ARM)** is a microprocessor architecture initially developed by a British company called Acron. The ARM company, a spin-off of Acron, licenses ARM technologies to semiconductor manufacturers and electronic device manufacturers.

ARM was originally conceived as a processor for desktop personal computers by Acorn Computers, a market now dominated by the x86 family used by IBM PC compatible computers. The relative simplicity of ARM processors made them suitable for low power applications. This has made them dominant in the mobile and embedded electronics market as relatively low cost and small microprocessors and microcontrollers. As of 2009, ARM processors account for approximately 90% of all embedded 32-bit RISC processors. ARM processors are used extensively in consumer electronics, including PDAs, mobile phones, digital media and music players, hand-held game consoles, calculators and computer peripherals such as hard drives and routers. The ARM architecture is licensable. Companies that are currently or formerly ARM licensees include Alcatel, Atmel, Broadcom, Cirrus Logic, Digital Equipment Corporation, Freescale, Intel (through DEC), LG, Marvell Technology

Group, NEC, NVIDIA, NXP (previously Philips), Oki, Qualcomm, Samsung, Sharp, ST Microelectronics, Symbios Logic, Texas Instruments, VLSI Technology, Yamaha and ZiiLABS.

ARM processors are developed by ARM and by ARM licensees. Prominent examples of ARM Holdings ARM processor families include the ARM7, ARM9, ARM11 and Cortex. Examples of ARM processors developed by major licensees include the DEC StrongARM, Freescale's i.MX, Marvell (formerly Intel) XScale, NVIDIA's Tegra, ST-Ericsson Nomadik, Qualcomm's Snapdragon, and the Texas Instruments OMAP product line as summarized in tables 1.1, 1.2, 1.3, 1.4.

In ARM-based machines, peripheral devices are usually attached to the processor by mapping their physical registers into ARM memory space or into the coprocessor space or connecting to another device (a bus) which in turn attaches to the processor. Coprocessor accesses have lower latency so some peripherals (for example XScale interrupt controller) are designed to be accessible in both ways (through memory and through coprocessors). In other cases, chip designers only integrate hardware using the coprocessor mechanism.

To improve compiled code-density, the Thumb mode was introduced. When in this mode, the processor executes 16-bit instructions. Most of these 16-bit-wide Thumb instructions are directly mapped to normal ARM instructions. The space-saving comes from making some of the instruction operands implicit and limiting the number of possibilities compared to the full ARM mode instruction.

Thumb-2 extends the limited 16-bit instruction set of Thumb with additional 32-bit instructions to give the instruction set more breadth (*i.e.* introducing bit-field manipulation, table branches, and conditional execution). A stated aim for Thumb-2 is to achieve code density similar to Thumb with performance similar to the ARM instruction set on 32-bit memory. In ARMv7 this goal can be said to have been met. The architecture has evolved over time, and starting with the v7 architecture three "profiles" are defined: the "A" (application), "R" (realtime), and "M" (microcontroller) profiles. To improve the ARM architecture for digital signal processing and multimedia applications, a few new instructions were added to the set. They are variations on signed multiply-accumulate, saturated add and subtract, and count leading zeros.

Family	Architecture Version	Core	Features	Cache (I/D)/MMU	Typical MIPS@MHz	Application
ARM7TDMI	ARMv4T	ARM7TDMI (-S)	3-stage pipeline, Thumb	none	15MIPS@16.8MHz, 63DMIPS@70MHz	Game Boy Advance, Nintendo DS, iPod, Lego NXT
StrongARM	ARMv4	SA-110		16KB/16 KB, MMU	203MHz 1.0DMIPS/MHz	Apple Newton 2x00 series, Acorn Risc PC, Psion Netbook
StrongARM	ARMv4	SA-1110	integrated SoC	16KB/16KB, MMU	233 MHz	Intel Assabet, Ipaq H36x0, Zaurus SL-5x00, Palm Zire 31
ARM9E	ARMv5TEJ	ARM926EJ-S	Thumb, Jazelle DBX, Enhanced DSP instructions	variable, TCMs, MMU	220MIPS@20 MHz	Mobile phones: Sony Ericsson (K, W series); Siemens and Benq (x65 series and newer); Texas Instruments OMAP1710, OMAP1610, OMAP1611, OMAP1612, OMAP-L137, OMAP-L138; Qualcomm MSM6100, MSM6125, MSM6225, MSM6245, MSM6250, MSM6255A, MSM6260, MSM6275, MSM6280, MSM6300, MSM6500, MSM6800; Freescale i.MX21, i.MX27

Table 1.1: ARM processor families and applications - part 1

Family	Architecture Version	Core	Features	Cache (I/D)/MMU	Typical MIPS@MHz	Application
Xscale	Armv5TE	PXA26x			default 400MHz, up to 624MHz	Palm Tungsten T3
Xscale	Armv5TE	PXA27x	Applications processor	32KB/32KB, MMU	800MIPS@624 MHz	Gumstix verdex,"Trizeps-Modules" PXA270 COM, HTC Universal, HP hx4700, Zaurus SL-C1000, 3000, 3100, 3200, Dell Axim x30, x50, and x51 series, Motorola Q, Balloon3, Trolltech Greenphone, Palm TX, Motorola Ezx Platform A728, A780, A910, A1200, E680, E680i, E680g, E690, E895, Rokr E2, Rokr E6, Fujitsu Siemens LOOX N560, Toshiba Portégé G500
Xscale	Armv5TE	Monahans		32KB/32KB L1, TCM, MMU	1000MIPS@1.25GHz	Samsung Omnia
Xscale	Armv5TE	PXA900				Blackberry 8700, Blackberry Pearl (8100)

Table 1.2: ARM processor families and applications - part 2

Family	Architecture Version	Core	Features	Cache (I/D)/MMU	Typical MIPS @ MHz	Application
ARM11	ARMv6	ARM1136J (F)-S	8-stage pipeline, SIMD, Thumb, Jazelle DBX, (VFP), Enhanced DSP instructions	variable, MMU	740@532-665 MHz (i.MX31 SoC), 400-528MHz	Texas Instruments OMAP2420 (Nokia E90, Nokia N93, Nokia N95, Nokia N82, Nokia N800, Nokia N810, Qualcomm MSM7200 (with integrated ARM926EJ-S Coprocessor@274 MHz, used in Eten Glofiish, HTC TyTN II, HTC Nike), Freescale MXC300-30 (Nokia E63, Nokia E71, Nokia 5800, Nokia E51, Nokia E75, Nokia N97, Nokia N81), Qualcomm MSM7201A as seen in the HTC Dream, HTC Magic, Motorola Z6, HTC Hero
ARM11	ARMv6KZ	ARM1176JZ (F)-S	8-stage pipeline, SIMD, Thumb, Jazelle DBX, (VFP), Enhanced DSP instructions	variable, MMU + TrustZone		Apple iPhone, Apple iPod touch, Conexant CX2427X, Motorola RIZR Z8, Motorola RIZR Z10, NVIDIA GoForce 6100[19]; Telechips TCC9101, TCC9201, TCC8900, Fujitsu MB86H60, Samsung S3C6410, S3C6430

Table 1.3: ARM processor families and applications - part 3

Family	Architecture Version	Core	Features	Cache (I/D)/MMU	Typical MIPS @ MHz	Application
Cortex	ARMv7-A	Cortex-A8	VFP, NEON, Jazelle RCT, Thumb-2, 13-stage superscalar pipeline	variable (L1+L2), MMU + TrustZone	up to 2000 (2.0 DMIPS/MHz in speed from 600MHz to greater than 1GHz)	Texas Instruments OMAP3xxx series, SBM7000, Oregon State University OSWALD, Gumstix Overo Earth, Pandora, Apple iPod touch (3rd Generation), Archos 5, FreeScale i.MX51-SOC, BeagleBoard, Apple iPhone 3GS, Motorola Droid, Palm Pre, Samsung i8910, Sony Ericsson Satio, Touch Book, Nokia N900, ZiiLABS ZMS-08 system on a chip
Cortex	ARMv7-A	Cortex-A9 MPCore	Application profile, (VFP), (NEON), Jazelle RCT and DBX, Thumb-2, Out-of-order speculative issue super scalar	, 1-4 core SMP MMU + TrustZone	2.5 DMIPS/MHz (per core)	Texas Instruments OMAP4430/4440, ST-Ericsson U8500

Table 1.4: ARM processor families and applications - part 4

Moreover, *Jazelle* is a technique that allows Java Bytecode to be executed directly in the ARM architecture as a third execution state (and instruction set) alongside the existing ARM and Thumb-mode.

The Advanced SIMD extension, marketed as NEON technology, is a combined 64- and 128-bit single instruction multiple data (SIMD) instruction set that provides standardized acceleration for media and signal processing applications. NEON can execute MP3 audio decoding on CPUs running at 10 MHz and can run the GSM AMR (Adaptive Multi-Rate) speech codec at no more than 13 MHz. It features a comprehensive instruction set, separate register files and independent execution hardware. NEON supports 8-, 16-, 32- and 64-bit integer and single-precision (32-bit) floating-point data and operates in SIMD operations for handling audio and video processing as well as graphics and gaming processing. In NEON, the SIMD supports up to 16 operations at the same time.

VFP (Vector Floating Point) technology is a coprocessor extension to the ARM architecture. It provides low-cost single-precision and double-precision floating-point computation fully compliant with the ANSI/IEEE Std 754-1985 Standard for Binary Floating-Point Arithmetic. VFP provides floating-point computation suitable for a wide spectrum of applications such as PDAs, smartphones, voice compression and decompression, three-dimensional graphics and digital audio, printers, set-top boxes, and automotive applications.

The ARM architecture is supported by Unix-like operating systems Linux, BSD, QNX, Plan 9 from Bell Labs, Inferno, Solaris, and iPhone OS. ARM is targeted to support wide range of browsers and OS's, from the classic Symbian and Windows Mobile to the newest Ubuntu, iPhone OS and Android.

The *XScale*, a microprocessor core, is Marvell's (formerly Intel's) implementation of the ARMv5 architecture, and consists of several distinct families: IXP, IXC, IOP, PXA and CE. Intel sold the PXA family to Marvell Technology Group in June 2006. The XScale architecture is based on the ARMv5TE ISA without the floating point instructions. XScale uses a seven-stage integer and an eight-stage memory super pipelined RISC architecture. It is the successor to the Intel StrongARM line of microprocessors and microcontrollers, which Intel acquired from DEC's Digital

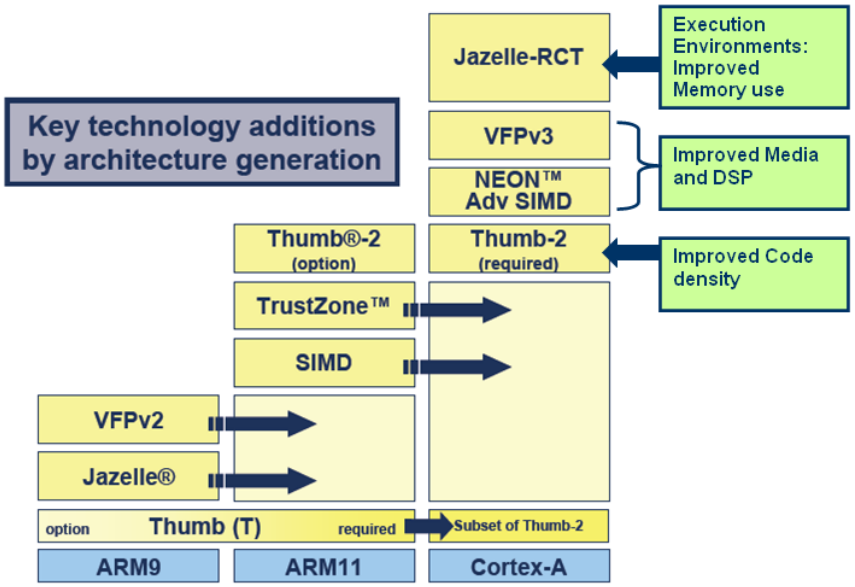

Figure 1.3: ARM processor evolution to Cortex

Semiconductor division as the side-effect of a lawsuit between the two companies. I

It consist on an ARM-compliant execution core with instructions ad data memory management units, data caches and buffers, power management, performance monitoring, debug and JTAGSunits, coprocessor interfaces, and core memory bus. The Xscale architecture features Intel dynamic voltage management technology, which allows operating voltage and frequency scaling on the fly.

XScale microprocessors can be found in products such as the popular RIM BlackBerry handheld, the Dell Axim family of Pocket PCs, most of the Zire, Treo and Tungsten Handheld lines by Palm, later versions of the Sharp Zaurus, the Motorola A780, the Acer n50, the Compaq iPaq 3900 series and many other PDAs. The XScale is also used in devices such as PVPs (Portable Video Players), PMCs (Portable Media Centres), including the Creative Zen Portable Media Player and Amazon Kindle E-Book reader, and industrial embedded systems.

On 2006 Intel agreed to sell the XScale PXA business to Marvell Technology Group; the move was intended to permit Intel to focus its resources on its core x86 and server businesses. Marvell holds a full Architecture License for ARM, allowing it to design chips to implement the ARM instruction set, not just license a processor core. The XScale effort at Intel was initiated by the purchase of the StrongARM division from Digital Equipment Corporation (DEC) in 1998. Intel still holds an ARM license even after the sale of XScale.

MIPS (originally an acronym for Microprocessor without Interlocked Pipeline Stages) is a reduced instruction set computer (RISC) instruction set architecture (ISA) developed by MIPS Computer Systems (now MIPS Technologies). The early MIPS architectures were 32-bit, and later versions were 64-bit. MIPS embedded processors and application-spcific integrated circuits (ASICs) are claimed to have the smallest silicon footprint and lower power consumption.

Through the 1990s, the MIPS architecture was widely adopted by the embedded market, including for use in computer networking, telecommunications, video arcade games, video game consoles, computer printers, digital set-top boxes, digital televisions, DSL and cable modems, and personal digital assistants. MIPS cores have been commercially successful, now being used in many consumer and industrial applications. MIPS cores can be found in newer Cisco, Linksys and Mikrotik's routerboard routers, cable modems and ADSL modems, smartcards, laser printer engines, set-top boxes, robots, handheld computers, Sony PlayStation 2 and Sony PlayStation Portable. In cellphone/PDA applications, the MIPS core has been largely unable to displace the incumbent, competing ARM core. The low power-consumption and heat characteristics of embedded MIPS implementations, the wide availability of embedded development tools, and knowledge about the architecture means use of MIPS microprocessors in embedded roles is likely to remain common.

Motorola/Freescale Semiconductor's DragonBall, is a microcontroller design based on the famous 68000 core, but implemented as an all-in-one low-power solution for handheld computer use.

The DragonBall's major design win was in earlier versions of the Palm Computing platform; however, from Palm OS 5 onwards it has been superseded by ARM-

based processors from Texas Instruments and Intel. The processor is also used in some of the AlphaSmart line of portable word processors. The processor is capable of speeds of up to 16.58 MHz and can run up to 2.7 MIPS, for the base 68328 and DragonBall EZ model. It was extended to 33 MHz, 5.4 MIPS for the DragonBall VZ model, and 66 MHz, 10.8 MIPS for the DragonBall Super VZ. It is a 32-bit processor with 32-bit internal and external address bus (24-bit external address bus for EZ and VZ variants) and 32-bit data bus. It has many built-in functions, like a color and grayscale display controller, PC speaker sound, serial port with UART and IRDA support, UART bootstrap, real time clock, is able to directly access DRAM, Flash ROM, and mask ROM, and has built-in support for touch screens.

The more recent DragonBall MX series microcontrollers, later renamed the Freescale i.MX series, are intended for similar application to the earlier DragonBall devices but are based around an ARM9 or ARM11 processor core instead of a 68000 core.

Memory and Storage

Memory represents another dimension of constraints for mobile devices, as it requires a small program footprint for both mobile operating system and applications. Three types of memory are used in this domain: RAM, ROM and flash memory. The latter is a special for of nonvolatile electrically erasable programmable read-only memory (EEPROM) that allows data to be read or erased on a block level, as opposed to the byte level of other ROM types. On desktop computers, physical memory is considered first-level storage, and hard drives (*i.e.* hard disk) are secondary storage. For mobile computer systems, because of the size of system code and because application codes are comparatively smaller than those of desktop computer systems, a pure first-level memory design can be implemented at moderate cost to achieve good performances. As a result, today's PDA and smart phones only use memory for code and data storage. Mobile devices often support I/O extension interfaces, thus allowing the use of large flash cards with gigabyte capacity. They allow faster access, they are smaller and lighter than hard disks, they are quiet and don't have mechanical parts, so they're preferred to hard disk for mobile devices. Moreover, hard disk ower consumption is prohibitively high, and data access speed does not satisfy the needs

of mobile applications.

Input device and Display

The input device is crucial to the adoption of a mobile device; available input devices used on cell phones, PDAs, smart phones and other handheld computing devices are:

- Cell phone keypad; it is a 12-button keypad consisting of keys from 0 to 9, each also representing some letters and characters and four function keys, *i.e.* call, hang up, menu, cancel.
- QWERTY keyboard. It is a tiny version of standard English computer keyboards or typewriters.
- Alphabetic keyboard. Another type of small keyboard, with keys arranged alphabetically.
- Stylus-based virtual keyboard and Handwriting Recognition. They are mostly popular on PDAs; a user can either use a stylus to write on the device screen or click keys on a virtual keyboard displayed on the screen. The stylus-based input method is well suited for frequent text input, but cannot be operated with one hand.

The display of mobile devices has evolved in a number of directions at a fast pace for many years. Display screen dimensions vary from device to device, predominantly determined by the purpose of the device. Smart phones, for example, often have a sufficiently large display screen for wireless Internet applications as well as for personal information management (PIM) applications. Design choices have to be made with regard to a number of factors:

- size - cell phones usually have 2.2-inch diagonal LCD screens with backlight. The screen size for PDAs range from 2 to 10 inches, while smart phones have screens of 3,7-4 inches.

- resolution - low resolution are still c.ommon among mobile devices; however, QVGA (320x240) but also 320x480 and 480x360 are supported for newer smartphones.

- color depht - it is very common to see consumer mobile devices with a color depht of 16 bit (65536 colors) or 18 bit (262144 colors)

- backlight - for a better display effect, the display screens of cell phones or PDAs usually have a backlight.

- power consumption - Thin-film transistor (TFT) displays tend to consume more power than earlier passive matrix displays (several hundreds milliwatts).

Actually, high performance smart phones use a touchscreen technology; there are two major categories for such a technology: capacitive and resistive. Capacitive touchscreens work by transferring a small electrical charge from the screen to your finger and detecting the region where the charge is removed. Resistive screens use two extremely thin layers below the glass that are pressed together when the screen is touched. The difference between the two means that resistive screens can be touched with any object, like a stylus, while capacitive screens need to make contact with your body, usually through your finger. In practice we find capacitive screens, like the one used on the iPhone, to be more responsive, though recent resistive screens, like the Sony Ericsson Satio, have shown that well designed resistive displays can be nearly as responsive. The second important element to consider for touchscreen devices is the design of the interface used by the manufacturers. Touchscreens demand that the icons are large enough to be pressed with a finger and well spaced enough to avoid accidentally pressing the icons beside it.

Because the display screen of a mobile device accounts for a significant amount of the power consumption of the mobile system, it must be considered in the overall power management scheme. In addition to employing advanced low-power display technologies, mobile software, including the operating systems, and applications, must be power aware and adaptively control the display.

1.2.2 Mobile software platforms and applications

Actually, the need for high performances in mobile devices involves the ability to browse any web site, handle mail and different kind of business document; manage complex user interfaces; support next generation 3D games; edit photos and videos and watch videos in any format; in addition to rich operating systems,and java and execution environment support.

For this reason, the whole mobile industry need to go through profound transformation from vertical to horizontal business model. It is necessary to adopt open interfaces either through standardization or through de facto industry standards with open specification - not forgetting the open source community with increasing exposure towards mobile platforms. Currently many inherent drivers in mobile device industry are pushing towards modular solutions, standardized interfaces, and utilization of third parties with special domain knowledge [7] A mobile software platform is defined as the combination of an operating system for a collection of compatible mobile devices with a set of related software development libraries, application programming interfaces (APIs) and programming tools. Mobile software platforms are either proprietary for special devices or open to all independent software providers. As mobile hardware technologies and wireless technologies continue to advance, mobile operating systems are required to take advantages of those improvements and provide strong support for application developers. In the following the major mobile platforms are presented.

Symbian

Symbian is a private, independent company that develops and supplies the open standard operating system Symbian OS designed for mobile devices and smartphones, with associated libraries, user interface, frameworks and reference implementations of common tools, developed by Symbian Ltd. It was a descendant of Psion's EPOC and runs exclusively on ARM processors. Symbian is owned by large cell phone manufacturers including Nokia, Ericsson, Sony Ericsson, Siemens and Samsung. Symbian OS is designed to support a wide range of voice and data services in 2G, 2.5G

and 3G cellular systems, as well as multimedia and data synchronization. Symbian OS was created with three systems design principles in mind: the integrity and security of user data is paramount, user time must not be wasted, and all resources are scarce. It is areal time, multithreaded, preemptive kernel that performs memory management, process and thread scheduling, interprocess communication, process relative and thread relative resource management, hardware abstraction and error handling. The basic services provide a programming framework for SymbianOS components, such as kernel and user API library, device drivers, file systems, and standard C++ library. On top of the basic services is a set of communication services, multimedia services, PC connectivity services and generic OS services as represented in figure 1.4. Communication services act as the core to mobile telephony and data network access applications. Personal area network (PAN) connectivity such as Bluetooth and IrDA, and USB is also enabled by these services. Multimedia services deal with audio and video recording, playback and streaming. Connectivity services are software components that implement PC synchronization. Generic OS services offer typical OS-related components such as memory management and file system access. Application services allow user programs to be executed in separated processes. The UI framework is comprised of an array of UI components and event-handling mechanism that allows easy porting of UI programs between different Symbian OS devices.

Furthermore, all Symbian programming is event-based, and the CPU is switched into a low power mode when applications are not directly dealing with an event. This is achieved through a programming idiom called active objects. Similarly the Symbian approach to threads and processes is driven by reducing overheads. Symbian OS uses EPOC C++, a pure object-oriented language, as the supporting programming language for both system services implementations and application programming interfaces. It also allows Java applications for mobile devices (J2ME apps) to run on top of a small Java runtime environment. The last version of Symbian OS is used devices as Samsung i8910 Omnia HD, Nokia N97, Nokia 5800 XpressMusic, Nokia 5530 XpressMusic and Sony Ericsson Satio.

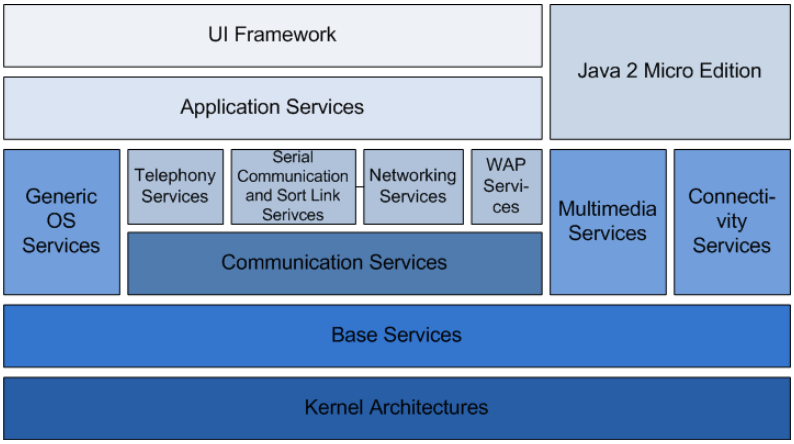

Figure 1.4: Symbian OS architecture

Palm

Palm OS (also known as Garnet OS) is a mobile operating system initially developed by Palm, Inc. for personal digital assistants (PDAs) in 1996; Palm Inc. is the company that created the PDA market.

Palm OS is designed for ease of use with a touchscreen-based graphical user interface and making use of limited computing power to allow efficient operations. It is provided with a suite of basic applications for personal information management. It has been implemented on a wide array of mobile devices, including wrist watches, handheld gaming consoles, barcode readers and GPS devices. Later versions of the OS have been extended to support smartphones.

Palm OS 6 (Cobalt) is a milestone in Palm OS history. It is a complete rwrite of previous versions and it is the first to support multithreading. It also provides more wireless capability, multimedia application support and a variety of extension slots. Palm OS allows third party hardware to be used as part of the system; the kernel of Palm OS is based on AMX, licensed from Kodak, which offers preemptive multitasking and protective memory management. System services are a set of modular components that provide communication, input method, GUI event handling and multimedia processing. Third party software libraries can also be plugged into

the system, for example J2ME profile implementations can be added as external libraries. Palm OS employs a database model to store files rather than a block-based file model. Recently Palm has released webOS, a mobile operating system running on the Linux kernel with proprietary components. webOS's graphical user interface is designed for use on devices with touchscreens. It includes a suite of applications for personal information management and makes use of a number of web technologies such as HTML 5, JavaScript, and CSS. The Palm Pre, released on June 2009, is the first device to run this platform.

Windows Mobile

Windows Mobile is a compact operating system developed by Microsoft, and designed for use in smartphones and mobile devices. It is based on Windows CE and is designed to support a variety of low-capability mobile devices such as PDAs, cell phones, smartphones, bar code readers, handheld computers and embedded devices in automobiles, among others. There are three versions of Windows Mobile for various hardware devices:

- Windows Mobile Professional runs on 'Windows Phones' (smartphones) with touchscreens
- Windows Mobile Standard runs on 'Windows Phones' with regular screens
- Windows Mobile Classic which runs on "Windows Mobile Classic devices" (Pocket PCs).

"Windows Mobile Classic device" is a Windows Mobile personal digital assistant (PDA) that does not have telephone functionality. It was formerly known as the Pocket PC. It was the original intended platform for the Windows Mobile operating system.

The .Net Compact Framework is loaded into the ROM of the underlying smartphone device. .Net is Microsoft's general software infrastructure of Internet-based computing. Its core is XML Web services. They are widely accepted as the solution to enable seamless, robust and secure collaboration among heterogeneous Internet

services and applications.What makes it possible to achieve this goal is the open-standard XML that is able to integrate data and its structure into a self-explanatory format such that they can be organized, edited, programmed, and exchanged between any applications, services, web sites and smart devices. The heart of the .Net infrastructure is .Net Framework on most Windows-based operating systems, including tablet PCs, Pocket PCs and Smartphones.

Embedded Linux

Linux is a free, open-source, UNIX-like operating system. It actually refers to a combination of two portions: Linux kernels and Linux applications. A vast number of free, open-source applications have been developed and are in use on various Linux systems. Source code of Linux kernels and applications are mostly available under Gnu Public License (GPL). Linux is diffused also in the embedded system market, powering up a broad range of network equipement, consumer electronics, industrial facilities and mobile devices. In particular, some Linux distributions for PDAs are free for download on the internet, and commercial Linux systems for mobile devices, such as Monta Vista Linux are available in the market. Embedded Linux systems could be either hard real time or soft real time. Hard real time means that the system must respond in a deterministic way every time a relevant event occurs. In soft real-time systems, quick responsiveness is desired but not guaranteed.Mobiledevices used by consumers generally fall into the latter category. The latest Linux kernels support both hard real time and soft real time applications due to the preemptive kernel design and the process scheduler.

Summarizing, the main characteristics of embedded Linux systems are:

- the monolithic kernel supports multitasking and multithreading and can be tailored for different applications scenario;
- the open-source community provides support for the latest new technologies, including Bluetooth, wireless LAN and wireless sensor network;
- application under GPL license can be modified and extended;

- license fee is low or nonexistent.

Several industry groups have formed to foster use of Linux in embedded applications. In particular, the Embedded Linux Consortium produced the ELCPS (Embedded Linux Consortium Platform Specification) which was intended as a guide to developers of embedded Linux devices as to what functionality should be included in order to provide a standard platform supporting application portability.

RIM Blackberry

RIM provides a proprietary multi-tasking operating system (OS) for the BlackBerry, which makes heavy use of the device's specialized input devices, particularly the scroll wheel or more recently the trackball. The OS is focused on easy operation and was originally designed for business , it provides support for Java MIDP 2.0 and WAP 1.2. Third-party developers can write software using proprietary BlackBerry APIs as well, but any application that makes use of certain restricted functionality must be digitally signed so that it can be associated to a developer account at RIM. This signing procedure guarantees the authorship of an application, but does not guarantee the quality or security of the code. The OS supports push e-mail, mobile telephone, text messaging, internet faxing, web browsing and other wireless information services as well as a multi-touch interface.

Modern GSM-based BlackBerry handhelds incorporate an ARM 7 or 9 processor, while older BlackBerry 950 and 957 handhelds used Intel 80386 processors. The latest GSM BlackBerry models (8100, 8300 and 8700 series) have an Intel PXA901 312 MHz processor, 64 MB flash memory and 16 MB SDRAM. CDMA BlackBerry smartphones are based on Qualcomm MSM6x00 chipsets which also include the ARM 9-based processor and GSM 900/1800 roaming (as the case with the 8830 and 9500) and include up to 256MB flash memory. The latest BlackBerry 9000 series is equipped with Intel XScale 624 MHz CPU,which makes the fastest BlackBerry to date. Several non-BlackBerry mobile phones have been released featuring the BlackBerry e-mail client which connects to BlackBerry servers.

iPhone os

iPhone OS, known as OS X or OS X iPhone in its early history, is the operating system developed by Apple for the iPhone and iPod touch. Like Mac OS X, from which it was derived, it uses the Darwin foundation. iPhone OS has four abstraction layers:

- the *Core OS* layer;
- the *Core Services* layer
- the *Media* layer
- the *Cocoa Touch* layer

The operating system takes less than 240 Megabytes of the device's total memory storage. iPhone OS' user interface is based on the concept of direct manipulation, using multi-touch gestures. Interface control elements consist of sliders, switches, and buttons. The response to user input is supposed to be immediate to provide a fluid interface. Interaction with the OS includes gestures such as swiping, tapping, pinching, and reverse pinching. Additionally, using internal accelerometers, holding the device sideways (so that the screen is in landscape orientation) alters the screen orientation in some applications. A home screen (rendered by "SpringBoard") with application icons, and a dock at the bottom of the screen, showing icons for the applications the user accesses the most, is presented when the device is turned on or whenever the home button is pressed. The screen has a status bar across the top to display data, such as time, battery level, and signal strength. The rest of the screen is devoted to the current application. There is no concept of starting or quitting applications, only opening an application from the home screen, and leaving the application to return to the home screen.

The central processing unit used in the iPhone and iPod Touch is an ARM-based processor (Cortex8) instead of the x86 (and previous PowerPC or MC680x0) processors used in Apple's Macintosh computers, and it uses OpenGL ES 1.1 rendering by the PowerVR 3D graphics hardware accelerator co-processor. Mac OS X applications cannot be copied to and run on an iPhone OS device. They need to be written

and compiled specifically for the iPhone OS and the ARM architecture. Authorized third-party native applications are available for devices with iPhone OS 2.0 and later through Apple's App Store.

Apple has not announced any plans to enable Java to run on the iPhone. The iPhone OS has been subject to a variety of different hacks for a variety of reasons, centered around adding functionality not supported by Apple, but it is not illegal. Before the SDK was released, third-parties were permitted to design "Web Apps" that would run through Safari. Unsigned native applications are also available. The ability to install native applications onto the iPhone outside of the App Store will not be supported by Apple. Such native applications could be broken by any software update, but Apple has stated it will not design software updates specifically to break native applications other than those that perform SIM unlocking.

The SDK itself is a free download, but in order to release software, one must enroll in the iPhone Developer Program, a step requiring payment and Apple's approval. Signed keys are given to upload the application to Apple's App Store. Applications can be distributed in three ways: through the App Store, through enterprise deployment to a company's employees only, and on an "Ad-hoc" basis to up to 100 iPhones. Once distributed through the App Store, a developer can request up to 50 promotional codes that can be used to freely distribute a commercial application he or she has developed. This distribution model for iPhone software appears to make it impossible to release software based upon code licensed with GPLv3.

Android

Android is a software stack for mobile devices that includes an operating system, middleware and key applications (figure 1.5). The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language. By providing an open development platform, Android offers developers the ability to build rich and innovative applications. Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, etc. Developers have full access to the same framework APIs used by the core applications. The

application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user. For example, an application can call upon any of the phone's core functionality such as making calls, sending text messages, or using the camera, allowing developers to create richer and more cohesive experiences for users. Android is built on the open Linux Kernel. Furthermore, it utilizes a custom virtual machine that was designed to optimize memory and hardware resources in a mobile environment. Android is open source; it can be liberally extended to incorporate new cutting edge technologies as they emerge. The platform will continue to evolve as the developer community works together to build innovative mobile applications.

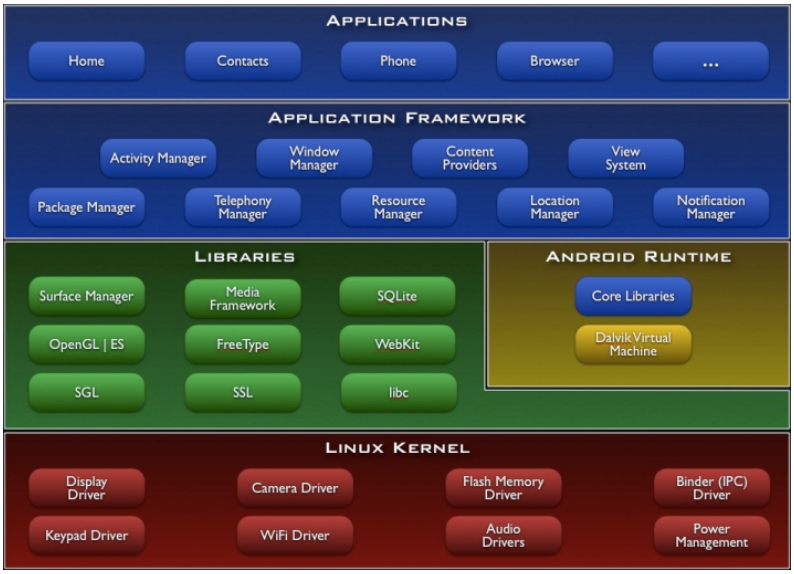

Figure 1.5: Android platform architecture

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:

- System C library - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based device
- Media Libraries - based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
- Surface Manager - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications
- LibWebCore - a modern web browser engine which powers both the Android browser and an embeddable web view
- SGL - the underlying 2D graphics engine
- 3D libraries - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer
- FreeType - bitmap and vector font rendering
- SQLite - a powerful and lightweight relational database engine available to all applications

It allows developers to write managed code in the Java language, controlling the device via Google-developed Java libraries Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

The first phone on the market to use the Android platform was HTC Dream, followed by HTC Hero, HTC MAgic, Motorola Droid, Samsung l7500 and l5700; HTC Google Nexus One is expected with the last version of Android platform (version 2.1).

Brew

Binary Runtime Environment for Wireless (BREW) is a wireless software platform solution for CDMA cell phones developed by QUALCOMM, the company that created the CDMA technology. BREW consists of three components: a binary runtime environment, an application developement environment and a distribution system. The binary runtime environment allows native BREW applicatins to operate regardless of the air interface used by the cell phone. The application development environment provides a set of platform APIs and supporting tools for BREW software development. The Brew distribution system (BDS) allows wireless data service providers to deliver content to subscriber. Any mobile applications that are about to be deployed on mobile device must be signed using a digital key and certificate. Brew is generally regarded as a hardware-independent wireless solution for mobile devices. In this sense it is competing with J2ME; these technologies are not entirely competitors, for example Qualcomm has selected IBM's J9 Java virtual machine for BREW, thus allowing to execute Java applications.

1.2.3 Java Micro Edition (J2ME)

Java is not a mobile operating system, but other mobile operating systems can leverage Java platforms to enable code portability and enforced security. In this sense Java, in particular J2ME, performs as a common middle layer between a specific mobile operating system and applications offered by wireless service providers.

Java by definition is a cross-platform, object-oriented, interpretive programming language developed by SUN Microsystems. At the heart of Java is a set of Java Virtual Machines (JVMs) from different platforms that interpret compiled Java bytecode. To run Java applications on a platform, a Java Runtime Environment (JRE) is required. A JRE consists of a JVM for the underlying platform, core classes in standard packages, and some supporting files.
The official Java platforms (JRE and software development kit) available from Sun Microsystems, each targeting a type of computing system are:

- Java 2 Standard Edition (J2SE), for desktop and server systems

- Java 2 Enterprise Edition (J2EE), for enterprise application systems
- Java 2 Micro Edition (J2ME), for embedded systems
- Java Card, for smart card applications

J2ME is aimed at the ever-expanding embedded devices market. Due to the diversity of devices in this segment, J2ME has ben furthered packaged into two distinct configurations: *Connected Limited Device Configuration (CLDC)* for mobile devices and *Connected Device Configuration (CDC)* for consumer and embedded devices.

A J2ME configuration consists of a set of fundamental requirements for JVMs and supportive Java classes and APIs that as a whole represent Java runtime environment for a collection of embedded devices with similar hardware and network capabilities.
The characteristics of CLDC devices can be summarized as follows:

- A 16-bit or 32-bit processor with a clock speed of 16 MHz or higher
- Low memory budget, 160 to 512 Kbyte
- Lower power consumption and limited power supply (mostly battery)
- Simple user interface, or no interface at all
- Low bandwidth and possibly intermittent network connection

Examples of CLDC devices include cell phones, low-end PDAs, pagers, wireless sensors, radio frequency identification (RFID) devices, etc. CLDC devices are resource constrained in terms of power, processing capability, memory capacity and network capability; therefore, standard JVM cannot be used. Instead, a stripped-down JVM, the Kilobyte virtual machine (KVM), is highly desirable for CLDC configuration. Sun has suplied a reference implementation of KVM along with the CLDC configuration. Other KVM implementations are also available, such as IBM's J9 virtual machine, which has been selected for the BREW platform by Qualcomm. The characteristics of CDC devices are:

- A 32-bit processor
- Large memory budget, 2MB RAM and 2.5 MB ROM available to the Java application environment
- Wired power supply or long battery time
- Stable network connection
- Various user iterface, from sophisticated GUIs to no UI

Examples include high-end PDAs, television set-top boxes, and RFID readers. Unlike CLDC, CDC configuration uses standard JVM. J2ME configurations are defined based on hardware and network specifications rather than vertical classification of application scenarios of various devices.. The functionality provided by devices of the same J2ME configuration may vary greatly, requiring additional APIs that are not provided by the underlying common-ground J2ME configuration. For this reason, some J2ME profiles have been defined for each configuration. A profile is a set of standard Java APIs for a specific narrower class of devices within a J2ME configuration. For the CDC configuration, a foundation profile, a personal basis profile and a personal profile are defined; the foundation profile ats as the core to the other two profiles. For the CLDC configuration, mobile information device profile (MIDP) and PDA profile have been devised. For cell phones and smart phones, the CLDC configuration and the MIDP profile made up a standard Java platform, as shown in figure 1.6.

1.2.4 Comparison between most diffused devices and platforms

Symbian OS and Palm OS are historically dedicated mobile operating systems for cell phones and PDAs, respectively. The challenge is to provide a wide range of mobile telephony and mobile data services with very limited resource on a small form factor. As an open source operating system, Linux has been embraced by a number of mobile device manufacturers as a low-cost and highly customizable solution. J2ME CLDC

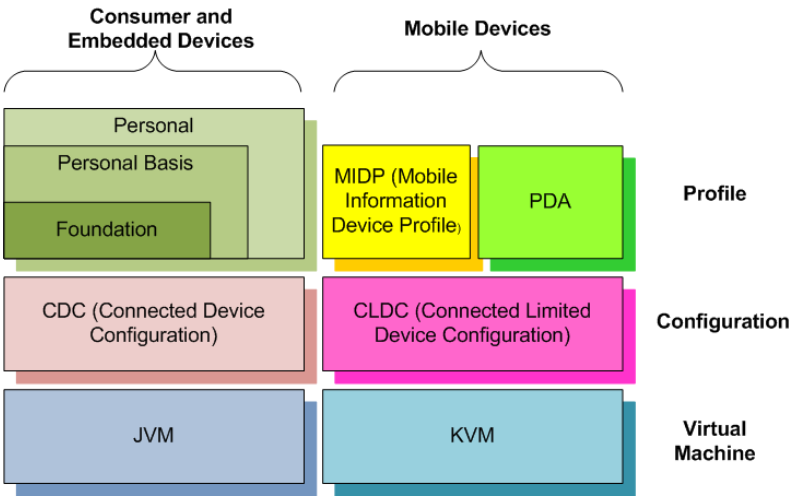

Figure 1.6: J2ME configurations and profiles

is platform independent and can be used on any mobile operating system supporting J2ME.

Microsoft Windows Mobile for Smartphone is rooted in the general purpose Windows CE operating system, and is known to require much more in the way of computing resources due to the generalized architecture of Win CE even though it has been tailored for smart phones. Microsoft approach, where the company supplies the software and general specifications, then leaves it to the handset makers to design and build the phones.

On the contrary, both Apple's iPhone and Research In Motion's BlackBerry are vertically integrated products, controlled from top to bottom by a single company. The advantages of integration are obvious. On handsets more than PCs, the success of a user interface depends on it being tailored to a specific piece of hardware. Every bit of the iPhone version of OS X seems tailored to the specific capabilities of the device, the keyboard and display in particular, which is a big reason why the user experience is so good. In the end, customers pay their money and make their choices. iPhone-like top-to-bottom will probably always yield the best integrated products. The price the user pay is that Apple gets to make all the choices. And in the case

of the iPhone, Apple also has control over what applications are approved to run on it. Apple has made several arguments in support of the strategy to control every aspect of its device, including the hardware (battery, screen, etc.) and the software (OS, applications, etc.) that runs on it.

The primary argument in Apple's support is that this way Apple can make sure that devices work the way they are expected to work. The other reason could be that Apple wants to prevent the loss of revenue for the thousands of developers developing applications for the App store (if you have a jailbroken phone, you can virtually install any application for free). In this sense, the relationship between Apple and the applications developers is a pure commercial one. The last argument is to ensure hardware compatibility; for hardware related issues, Apple may want to ensure that the hardware is capable to support the software. iPhones are extremely powerful devices (for their size) and the described philosophy may induce to think that Apple is preventing users to take full advantage of its powers. Instead of making their devices/OS strong to handle such uses, Apple has chosen an easier route to allow only certain things on its devices so as to avoid the trouble of making the device more secure.

The arrival of a much more mature Android (version 2.0 on the Motorola/Verizon Droid and 2.1 on expectes HTC Google Nexus One) means we are going to see a fair fight for the future of smartphones between the models of vertical integration and open platforms. Android has a real chance because it started out as a much more modern design and Google has shown the ability to evolve in Internet time. In a little over a year, we have seen two major versions of Android, with two significant point releases (1.5 "Cupcake" and 1.6 "Donut") in between. Still there are risks in the platform approach. One is that Google has relatively little control over how hardware makers use the open source Android. The biggest risk would be what's known in the software development business as a code fork, which means that we would see different versions of Android that would not all run the same applications. In this case, Google will use its heft and influence to keep this from happening. But even relatively simple variations, like handsets with or without physical keyboards, can lead to software compromises the produce less-than-optimal experiences.

Things are a little rougher in BlackBerry-land, where RIM has to support devices with more diverse capabilities. The touchscreen Storm2 in particular has put a strain on the one-size-fits-all OS; some features designed for keyboard-equipped BlackBerrys don't feel right on the big touchscreen. Still, RIM has also done a very good job of delivering a very good experience.

Windows Mobile until now has been the leading platform choice (Symbian is a contender), but it has some problems that the newest version 6.5 has not solved yet; new touchscreen phones come with a stylus, which immediately takes us back a decade or more. Worse, the stylus isn't just an atavism; with the insensitive resistive touchscreens that WinMo still requires, there are times when user really need it. An interim release that will support more modern capacitive displays is due in a few months, but a major overhaul, Windows Mobile 7, is not expected to show up in handsets until a year from now.

Today, with iPhone, BlackBerry, and Android, we have three world-class product families with different approaches to software and hardware development. BlackBerry tends to live mainly in its own email-centric world, but iPhone and Android will be going at it head on. The developers have deserted Windows Mobile and are now writing applications for Android and iPhone instead. Hence, Windows Mobile has less than 300 applications in the application Market Place. While Windows CE has a lot of legacy applications from long ago, Windows Mobile has very few new applications, because no new development is happening. Microsoft will release Windows Mobile 7 next year, which will bring multi-touch gestures and support for capacitive screens at an OS level. This update will catch up to where competitors were 3 years ago. But because it still will have few applications, it will gain little on what Windows Mobile 6.5 is today. The inevitable result will be that Windows Mobile is discontinued for consumer smartphones. However, Windows CE derivatives will still live on, but in specialized industrial settings only.

In the following a summarizing table of smartphones that in 2009 offered the perfect mix of features, performance and value for money (figures 1.5, 1.6).

	 HTC Hero	 Samsung HD-Icon	 Apple iPhone 3GS
Battery	8h 750 h standby		5 h 300 h standby
Camera	5-megapixel 4x	8-megapixel 4x	3-megapixel
Connectivity	Bluetooth, 3G, HSDPA Data: GPRS, WAP, UMTS, HSDP	Bluetooth, Wi-Fi, 3G, HSDPA Data: GPRS, EDGE, HSDPA	Bluetooth, Wi-Fi (802.11b/g) Data: GPRS, WAP, EDGE, UMTS, HSDPA
Display	320 x 480 pixels 65k colours	360 x 640 pixels 16 million colours	320 x 480 pixels
Memory	800 MB microSD (exp. slot)	8GB microSD (exp. Slot)	32GB
OS	Android	Series 60	iPhone OS
Input method	Touchscreen	Touchscreen	Touchscreen
Dim. WxDxH	56.2 x 14.4 x 112 mm	58 x 12.9 x 123 mm	62 x 12 x 115 mm
Video and picture	MPEG-4, WMV JPG	MPEG-4 JPG	MPEG-4 JPG

Table 1.5: Comparison between smartphones - part 1

	 **nokia N900**	 **Blackberry Bold 9700**	 **Sony Ericsson Satio**
Battery		6 h talk 500 h standby	4.5 h talk 340 h standby
Camera	3-megapixel 4x	12Mpixels 4x	3-megapixel
Connectivity	Bluetooth, Wi-Fi (802.11b/g) Data: GPRS, WAP, EDGE, UMTS, HSDP	Bluetooth, Wi-Fi (802.11b/g) Data: GPRS, WAP, EDGE, UMTS, HSDPA	Bluetooth, Wi-Fi, 3G, HSDPA Data: GPRS, EDGE, UMTS, HSDPA
Display	800 x 480 pixels 16 million colours	480 x 360 pixels 65k colours	640 x 360 pixels 16 million colours
Memory	256 MB RAM 32GB ROM microSD (exp slot)	256 MB ROM 2GB memory card Included microSD (exp. slot)	8GB microSD (exp slot)
OS	Linux (on ARM Cortex-A8)		Symbian
Input method	QWERTY keyboard, Touchscreen	QWERTY keyboard	Touchscreen
Dim. WxDxH	60 x 18 x 111 mm	60 x 14 x 109 mm	54 x 15 x 111 mm
Video and Picture	MPEG-4, WMV JPG	MPEG-4 JPG	MPEG-4, Real Video, WMV JPG

Table 1.6: Comparison between smartphones - part 2

1.3 Service oriented infrastructures for pervasive computing

To develop pervasive services and applications, research in engineering methodologies and software architectures has focused service discovery and composition. In a ubiquitous computing environment, a service-oriented infrastructure must be enabled with service discovery protocols (SDPs) to find the most appropriate services, either upon direct request from the users or proactively. Moreover, mobility and resource scarcity introduce two dimensions that service-oriented infrastructures for wired networks don't take into account: location awareness and physical proximity between the service provider and the user. In a broader vision, to find the most appropriate services, the service-oriented infrastructure should exploit context. To improve decentralization, scalability, robustness, and to avoid single points of failure, the peer-to-peer paradigm is a viable solution for such advanced service-oriented infrastructures. In contrast with the client/server approach, in which resource providers and resource consumers are clearly distinct, peers usually play both roles. The key concept of the peer-to-peer paradigm is leveraging idle resources to do something useful, like cycle sharing or content sharing. In [8], Bodhuin et al. compare some traditional solutions for net-centric computing middleware, such as Jini, OSGi and CORBA, listing their pros and cons. The survey does not include Sun MicroSystem's JXTA [9], probably due to the fact that in year 2005 an implementation for mobile devices was not completed. JXTA is mainly the specification of a set of open protocols for building overlay networks, independent from platforms and languages. Currently there are three official implementation of JXTA protocols: J2SE-based, J2ME-based and C/C++/C -based. In particular, an almost complete version of the JXTA Java Micro Edition (JXTA-J2ME, a.k.a. JXME) has been recently released. It provides a JXTA compatible platform on resource constrained devices using the Connected Limited Device Configuration (CLDC) with Mobile Information Device Profile 2.0 (MIDP), or Connected Device Configuration (CDC). Supported devices range from smartphones to PDAs.

1.3.1 Ubiquitous peer-to-peer sharing of services

In this context, Web Service technologies provide standard, simple and lightweight mechanisms for exchanging structured and typed information betweens services in a decentralized and distributed environment. Web Services are a text-based Machine-to-Machine (MMI) interface, they are simple to build and take advantage of the ubiquity of the Web. The Web service community has addressed a number of issues in the context of Internet such as developing languages to describe services and the design of business processes by combining Web services.

Orchestration and choreography standards of Web services workflows address both the language for describing the process workflow and the supporting infrastructure for running it. OSGi [10] is a Java-based technology which provides a service-oriented plug-in-based platform for application development. The core component of the OSGi Specifications is the OSGi Framework, which provides a standardized environment to applications (called bundles). On top of the Framework, services are specified by a Java interface. Bundles can implement this interface and register the service with the Service Registry. Clients of the service can find it in the registry, or react to it when it appears or disappears. Advanced networking features, such as *e.g.* peer-to-peer connectivity, are not provided by OSGi and must be implemented on top of it.

The Web Service community considers services as the only mean for accessing resources (this concept has been explicitly formalized in the WSRF specification [11]), yet centralized registries, themselves exposed as services (like UDDI), are still deemed the primary tool to support the publication and the discovery phases. Unfortunately, a peer-to-peer network of Web Service providers with a publication/discovery infrastructure implemented as a set of interacting Web Services would be absolutely unefficient due to the heaviness of the SOAP messaging protocol.

On the other side, in JXTA each peer's service is just an example of resource which can be exploited by the user which owns the peer, or shared in the network, *i.e.* advertised by the user and exploited by other users. Resource descriptions have the shape of XML documents, namely advertisements. A JXTA advertisement can be filled with any document, *e.g.* a WSDL interface if the shared resource is a Web

Service. JXTA provides a lot of flexibility by separating basic infrastructural services, mandatory for all peers, from specialized services, with different levels of description and efficiency.

Within the context of JXTA and Web Service integration, Distributed System Group (DSG) of University of Parma is responsible for the development and maintenance of the JXTA-SOAP component [12], enabling Web Service deployment in JXTA peers, as well as distributed WSDL publication and discovery, and SOAP message transport over JXTA pipes (*i.e.* virtual communication channels which may connect peers that do not have a direct physical link, resulting in a logical connection bound to peer endpoints corresponding to available peer network interfaces with an example being a TCP port and associated IP address). JXTA-SOAP is currently implemented in two versions: J2SE-based (fully featured, extending JXTA-J2SE) and J2ME-based (partially featured, extending JXME), and is the sole open source project for P2P sharing of Web services being actively maintained and updated.

WSPeer [13] is a J2SE toolkit for deploying and invoking Web Services in peer-to-peer Grid environments, which wraps Globus Toolkit core libraries to support the WS Resource Framework (WSRF) [11]. More interesting for ubiquitous computing environments is the Mobile Web Services Mediation Framework (MWSMF) [14, 15], an adaptation of Apache ServiceMix, which is an open source ESB (Enterprise Service Bus). It provides an hybrid solution, since it must be configured as JXTA-J2SE peer and established as an intermediary between Web Service clients and mobile hosts, the latter being configured as JXME peers. Web Service clients can invoke the services deployed on mobile hosts via the MWSMF, which compresses SOAP messages (to BinXML format) and sends them through JXTA pipes. The MWSMF also manages message persistence, guaranteed delivery, failure handling and transaction support. Unfortunately, the source code is not publicly availble and few details are given about the realization of lightweight Web Service providers running on mobile hosts.

1.3.2 Web Services on resource-constrained devices

The ubiquity of Web Services makes them interesting to use for mobile applications. Device proliferation should not cause protocol proliferation; for this reason, using Web Services makes much sense for distributed mobile applications. Besides hardware constraints, mobile devices introduce many other specific challenges which make difficult the deployment of Web Services on top of them [16]. Unlike dedicated servers, mobile devices will typically have intermittent connectivity to the network. As a result, the services offered on a mobile device may not be accessible all the time. An application that uses or composes such Web Services needs to operate in an opportunistic manner, leveraging such services when they become available. On the server side, Web Services on mobile devices should also attempt to keep messages as short as possible. Another issue to be addressed is the change of IP address which may arise when a mobile device moves between different locations, and from one administrative domain to another. However, with the P2P in place, the need for the Public IP can be eliminated and the mobiles can be addressed with unique peer ID. Each device in the P2P network is associated with the same peer ID, even though the peers can communicate with each other using the best of the many network interfaces supported by the devices like Ethernet, WiFi, etc. [23]. Web Services and mobile applications can be related in three ways as represented in figure 1.7:

- Web Service Proxy: the backend of a mobile system uses Web Services to retrieve information and return it to the mobile user. The interface between the back-end system and the device remains consistent and can be implemented through whatever communication protocol may be necessary.

- Direct Connection to Web Services: mobile devices can directly access the network through the use of Web Services, thus avoiding the use of a proxy.

- Mobile devices as Web Service providers: more powerful performing mobile devices can bacome service providers hosting a lightweight web server and deploying services.

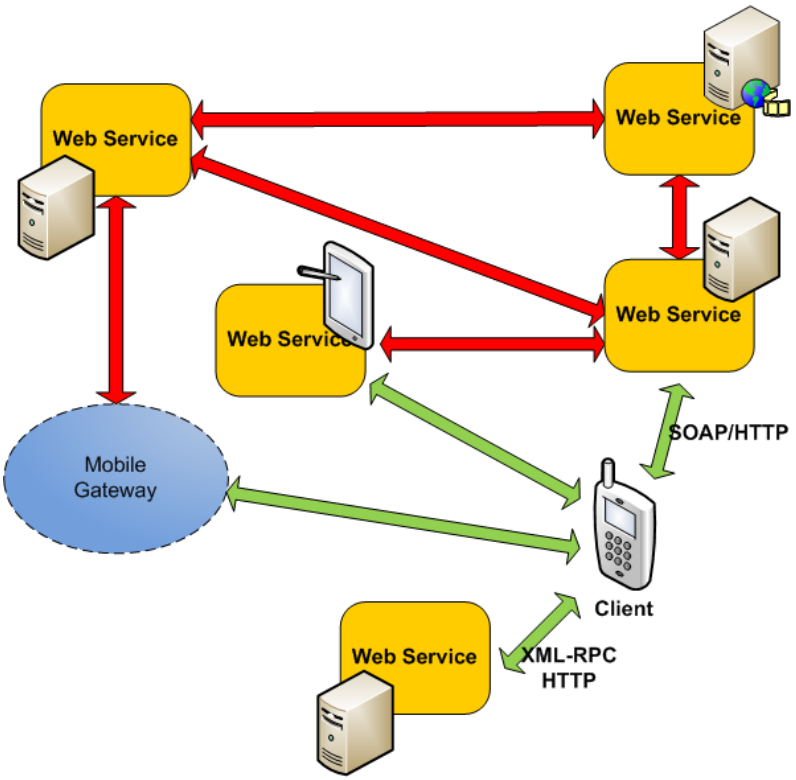

Figure 1.7: Web Services for the mobile infrastructure

Since the WS message protocol, namely SOAP, introduces some significant overhead, few toolkits support the deployment of Web Services on limited devices, such as PDAs, smart phones, etc. One is gSoap [17], which provides a WS engine with runtime call de-serialization. Unfortunately, gSoap is written in C/C++, thus requiring a priori stub/skeleton generation by means of a specific compiler, which also means lack of portability.

.NET Compact Framework [18] is a subset of the .NET platform, targeting mobile devices. Its class library enables the development of Web Service clients, but does not allow to host Web Services.

Looking at the Java Micro Edition (J2ME) platform, most libraries are only for client side functionality. The Java Wireless Toolkit (WTK) provides J2ME Web Services API (WSA) [19], based on JSR 172 [20], which specifies runtime Service-Provider interface to allow the generation of portable stubs from WSDL files. The specification contains some notable limitations, most of them due to the requirement for WS-I Basic Profile compliance. Conforming to the profile ensures interoperability, but also prevents using alternative methods. Another widely used solution is the kSoap2 [21] open source component, which is a parser for SOAP messages (with RPC/literal or document/literal style encoding), not supporting the generation of client side stubs. kSoap2 is compliant with devices lacking JSR 172 support, and allows to access non WS-I conformant services. To the best of our knowledge, the unique solution enabling J2ME applications (CLDC, CDC) as service endpoints is the Micro Application Server (mAS) [22]. It can be considered a lightweight version of Axis, by which it is inspired.

Chapter 2

Framework

In this chapter we introduce the formal framework we contributed to define, during the Ph.D. period, in the context of DSG research and development activity. The framework, called NAM, can be used to design distributed computing systems with shared resources. Here we introduce also NSAM, which is the service-oriented version of NAM, and we use it to address the service composition problem. Finally, we discuss about code mobility aspects, that we used to complete NSAM with a service mobility framework.

2.1 Networked Autonomic Machine

A Networked Autonomic Machine (NAM) is a hardware/software entity that is programmed to be completely altruistic, providing resources to other NAMs. In a peer-to-peer system of NAMs, each node can act both as resource consumer and resource provider, and contributes to the effective and efficient functioning of the whole system. NAMs can be of different types and complexities, depending on the device and on the characteristics of the offered resources. Several kinds of devices are considered: PCs and workstations, notebooks, PDAs, smart-phones, as well as sensors and actuators. Devices can be classified on the basis of their system characteristics (OS, processor type, memory, I/O type, battery, connectivity) or their functionalities

(camera, communicating, processing, sensors...). The software layer of each NAM includes a lightweight control module implementing a peer-to-peer overlay scheme.

Formally, a NAM node is a tuple

$$NAM =< UID, CTRL, R > \tag{2.1}$$

where UID is a unique identifier, $CTRL$ is the control layer, and R is a set of resources.

The control system is defined as

$$CTRL =< X, Y, S, S_R, \delta_{int}, \delta_{ext}, \lambda, t_a > \tag{2.2}$$

where

- X is is the set of input values
- Y is the set of output values
- S is a set of states
- S_R is the set of states of the resources owned by the NAM
- $\delta_{int} : S \times S_R \rightarrow S$ is the internal transition function
- $\delta_{ext} : Q \times X \times S_R \rightarrow S$ is the external transition function, where $Q = \{(s, t_e) | s \in S, 0 \leq t_e \leq t_a(s)\}$ is the total state set (t_e is the time elapsed since last transition)
- $\lambda : S \rightarrow \mathbb{R}^+$ is the output function
- $t_a : S \rightarrow Y$ is the time for which the system stays in state S if no external event occurs

The external transition function dictates the system's new state when an external event occurs. This state is determined by the input, the current state and how long the system has been in this state.

Resource attributes describe the device characteristics; some of them may change with time, others are fixed:

$$R = \{r_1, r_2, ..., r_n\} \tag{2.3}$$

Example of resource:

$$r1 = devicehardware =< CPU, memory(t), battery(t), connectivity >$$

representing the hardware of the device. Of course, resources can be defined with finer granularity. Each resource property has a name and a range. In the example, battery is represented by a percentage, connectivity is a string in $wired, wireless$ or in a more rich enumeration, CPU is an integer value, etc. Some properties are time-dependant.

2.1.1 Services as NAM resources

A service is a resource consisting in a unit of work executed by a service provider to achieve the results desired by a service consumer. Formally, a service is a tuple

$$\sigma =< I, O, P, E, C > \tag{2.4}$$

where I is a set of input parameters, and O a set of output parameters. Each I/O parameter has a type, i.e. a class (still using the ontological terminology). It is important that service consumers and service providers share the same domain ontologies in order to have a common understanding of shared services. Semantic descriptions of services are used to organize service advertisements in centralized or distributed repositories, allowing to efficiently retrieve and use services in the network. P and E are the precondition and effect sets, respectively. Such optional parameters are expressed in the form of logical conditions which can assume the true or false value. Preconditions must be verified in order to invoke the service, while an execution effect may become a precondition for the successive invocation in a composition scenario. For example, in an ambient intelligence scenario, if we need a service that assigns the value *"ON"*

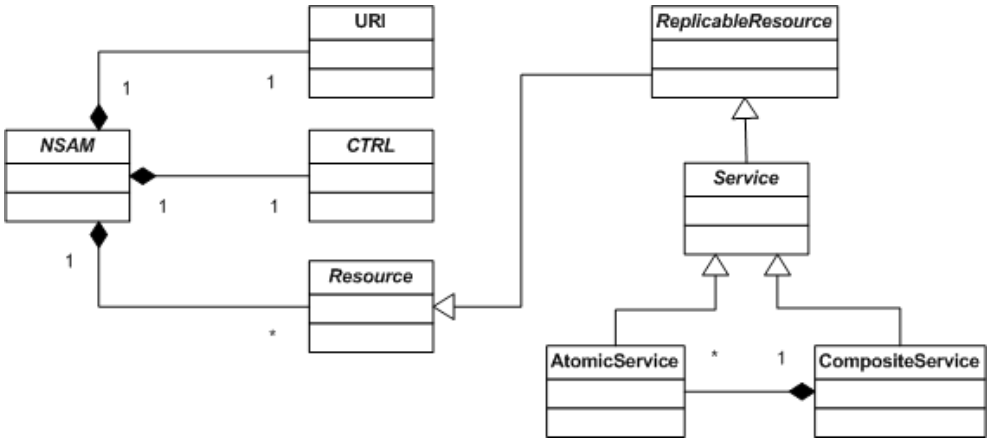

Figure 2.1: NSAM basic ontology.

to the *"status"* property of a *"living room light"*, we specify an invocation effect very precisely. C is the set of classes associated to the service.

The service-oriented version of NAM is called *Networked Service-oriented Autonomic Machine (NSAM)*. In NSAM network, each NSAM provides atomic services and cooperates with other NSAMs to build composite services.

An atomic service is defined as the minimal executable function unit, that cannot be decomposed and whose execution can transform a given state to another state. It is represented as a tuple:

$$a = < I, O, P, E, Q > \tag{2.5}$$

where Q is the set of quality of service attributes, depending on the device characteristics and on the amount of resource required to process inputs and generate outputs. Each node can provide different atomic services. The number of concurrent service instances and the quality of service (QoS) of each instance at a certain time depends on the current availability of hardware resources on the node.

Atomic services provided by different peers can be statically or dynamically aggregated (proactively or on-demand) to realize new complex tasks. A composite ser-

vice is a tuple:

$$c =< I, O, P, E, Q, G_w > \tag{2.6}$$

where G_w is the rule that allows to combine atomic services; this rule is represented as a directed workflow graph

$$Gw =< R_w, L_w > \tag{2.7}$$

where R_w is a set of services (both atomic and composite) and L_w is a set of links that represent transitions (i.e. I-O connections) among services.

2.1.2 Service composition

In a service-oriented infrastructure, user and application requests typically need to combine the functionality of several services and resources spread over the networked environment. The mechanism of combining two or more services together to form a complex service is known as service composition. Typically, a service composition system accepts a complex user task as an input and attempts to meet the needs of the task at hand by appropriately matching the task requirements with the available services. Such composite services enable users (applications) to reach their goal without having to discover and coordinate among a number of services on their own.

According to the NSAM model, a composite service is defined as aggregation of atomic and composite services (recursion). This allows the definition of increasingly complex applications by progressively aggregating components at higher levels of abstraction.

Creating a complex process requires not only a clear definition of collaboration patterns of all its components, but also a way of depicting service interactions. Task resolution is performed firstly deriving several different compositions at the semantic level, then identifying the underlying services that can take part in the composite results. Service composition mechanisms are classically treated as extensions to service discovery strategies (that are usually implemented as atomic services hosted by all peers). Service discovery is achieved by matching service requests with the

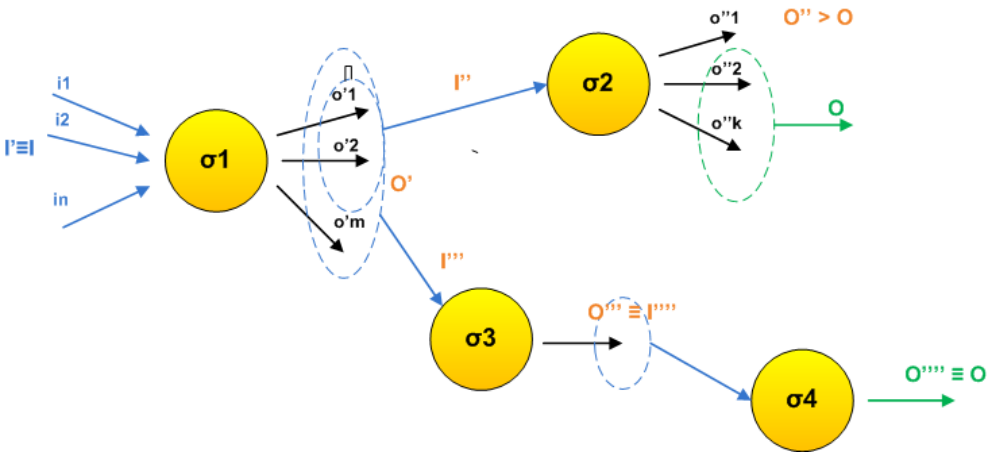

Figure 2.2: Example of service compositions.

ontology-based service descriptions of shared services. According to the proposed NSAM model, I and O attributes are used as parameters for discovery mechanisms. When a peer receives a service request that cannot process by itself, either partially or completely, it searches for other peers able to process the request. For this reason, it should be able to locate peers that provide any type of service and to send messages to a fraction of its neighbours in order to propagate the requests.

The requirements for service composition are that the output produced by a service σ_1 can be consumed by σ_2, i.e. for each $o \in O_1$, $\exists i \in I_2$, so that $type(o) = type(i)$ and $sem(o) = sem(i)$.

Figure 2.2 illustrates an example in which a service with input set I and output set O is composed using two alternative, semantically matching, flows:

$\sigma_1 \rightarrow \sigma_2$

since $I \equiv I1, O1 \equiv I2, O2 \equiv O$

$\sigma_1 \rightarrow \sigma_3 \rightarrow \sigma_4$

since $I \equiv I1, O1 \equiv I3, O3 \equiv I4, O4 \equiv O$

The quality parameter Q in the definition of the requested service is used to select a composition among all the possibilities, and to stop the discovery process when at least a composition with the required quality of service is discovered.

A service providing system is considered self-adaptive if it can dynamically adjust its service structure so as to reflect the changing demand and improve user's satisfaction. To make a system self-adaptive an effective coordination mechanism can be created in which peers are considered to be cooperative in nature.

Some NSAMs may also act as orchestrators for service composition, offering a service that collects service information (by triggering discovery processes), and creates a combination of available services that meet user requirements. Such coordinators manage both the discovery process and the service invocation once a satisfying composition is found.

Finally, one of the major challenges in pervasive computing applications is the issue of mobility. In any pervasive computing environment, once the initial composition is identified and a service session is established, the mobility of the peer can change the composed solution. In such situations, the challenge is to reconfigure the session under progress as quickly as possible by considering the current resource availability around the user.

2.1.3 Related works

Before developing our own framework, we studied state-of-art solutions, such as Gator Tech [23], Amigo [24], Socam [25].

Among others, the most innovative is PERSONA (Perceptive Spaces prOmoting iNdepentent Aging) [26], which is a EU-funded research project (FP6) started in 2007, aiming at developing a scalable open standard technological platform to build a broad range of Ambient Assisted Living (AAL) Services. In this context the main technical challenge is the design of a self-organizing middleware infrastructure allowing the extensibility of component/device ensembles in an ad hoc fashion. In order to achieve this goal the communication patterns of the infrastructure must be able to execute distributed coordination strategies in order to provide the necessary service discovery, service orchestration and service adaptation functionalities. The components of a PERSONA system are interfaced with the PERSONA middleware that enables the allocation of a different number of communication buses, each of them adopting specific and open communication strategies. Components linked with the

PERSONA middleware may register with some of these communication buses, find each others and collaborate trough the local instances of the buses. Input and output buses support multi-modal user interactions with the system. The context bus is an event-based channel to which context sources are attached, in particular the Wireless Sensor Networks (WSN) [27] are attached to this bus. Published events may be re-elaborated and transformed in high level events (situations) by components that have subscribed to the bus (e.g. context reasoners). The service bus is used to group all the services available in the AAL-space, being them atomic or composite (whose availability is managed by a Service Orchestrator component). Services belonging to the service bus may be requested by the Situation Manager in consequence of situation detections and rules stored in the Knowledge Base of the system. Generally, devices are attached to both the context and service bus. The former is used to send notifications of status changes, the latter to answer to status query or execute actions (e.g. switch on the light device). Many other basic components are foreseen in PERSONA system, but their discussion is out of the scope of this thesis (see [28] for details).

With respect to PERSONA, our NAM framework does not distinguish between software and hardware components. Moreover, resources (and in particular services) are managed according to distributed algorithms, rather than centralized components like the Service Orchestrator.

The *Pervasive Information Community Organization (PICO)* [29] is a middleware framework that enhances existing Internet-based services, creating mission-oriented dynamic computing communities that perform tasks for users and devices. It consists of autonomous software entities called *delegents* (or intelligent delegates) and hardware devices called *camileuns* (namely connected, adaptive, mobile, intelligent, learned, efficient, ubiquitous nodes) creating communities of delegents that collaborate proactively to handle dynamic information, provide selective content delivery, and facilitate application interface. A device model captures the characteristics and features of the available hardware resources. The identified features in the device model are provided as services over the network by creating software entities called delegents (intelligent delegates). Many such delegents can be combined together into a cooperative structure called communitiy. The communities offer a transparent ser-

vice usage mechanism within the PICO framework. Moreover, delegents representing low-resource devices can carry out tasks remotely. Camileuns and delegents are PICO's basic building blocks. Delegent can gather information locally or remotely to collaborate with other delegents to form a computing community. After completing the designated tasks in the remote camileun, a delegent returns to the active state. A delegent may terminate itself after achieving its goal or mission.

A dynamic service composition mechanism [30] has been implemented in PICO framework. The descriptions of discovered services are stored in a centralized directory, where also resources are exported as services. Users (applications) approach the directory to locate one or more services they need. The directory performs a query on the registered services and returns a matching service if found. By employing semantics, formal declarative descriptions are attached to services. Computer programs can use these descriptions to find the appropriate services and use them correctly. Such an architecture can typically be mapped onto managed environments with well-defined nodes in the network that can act as directories.

The PICO approach to dynamic service composition is similar to NAM's, except for PICO use of centralized directory for service discovery. On the contrary, PICO introduces the concept of community, as a group of collaborating delegents that is created for a specific task and exists for a limited period of time, while cooperation in NAM is enabled through an always available P2P network. Moreover NAM is a hardware/software entity, whose characteristics depend on the hosting device capabilities, while PICO delegents and camileuns (*i.e.* software and hardware entities) are completely separated entities.

SpiderNet framework [31] executes a novel bounded composition probing (BCP) protocol to provide fully decentralized QoS-aware service composition. SpiderNet leverages on a service-oriented P2P system called P2P service overlay where peers can provide not only media files but also a number of application service components such as media transcoding and data filtering as well as application-level data routing. P2P service overlays promote Internet-scale service sharing without any administration cost or centralized infrastructure support. New services can be flexibly composed from available service components based on the user's function and quality-of-

service (QoS) requirements. SpiderNet supports directed acyclic graph composition topologies and explores exchangeable composition orders for enhanced service quality. During service runtime, SpiderNet provides proactive failure recovery to overcome dynamic changes (*e.g.*, peer departures) in P2P systems. The proactive failure recovery scheme maintains a small number of dynamically selected backup compositions to achieve quick failure recovery for soft realtime streaming applications. The backup compositions are adaptively selected based on the conditions of the current composition and the user's QoS requirements. Thus, they can avoid the delay and overhead of triggering BCP to find a new composition if one of the maintained backup compositions can recover the failure. The SpiderNet system is implemented as a distributed middleware infrastructure deployed in wide-area networks, which can automatically map the user's composite service request into an instantiated distributed application service in the P2P service overlay. SpiderNet decentralized service discovery based on the Pastry distributed hash table (DHT) system. When a peer wants to share a service component, it registers the service component by storing the component's static meta-data (*e.g.*, location, input QoS, output QoS) into the DHT system.

The SpiderNet framework does not target mobile and resource-contrained devices, only considering a peer-to-peer network of static desktop PCs. NAM, on the contrary is designed with respect to a very dinamically changing environment, where not only peers can abrubtly leave the network, but the QoS of the services they provide may vary with the available resources of hosting devices.

CARMEN (Context Aware Resource Management ENvironment) [32] is a middleware for context-aware resource management, capable of supporting the automatic reconfigurationo Web Services for the wireless Internet in response to context changes, without any intervention on the service application logic. CARMEN allows service providers, system administrators and final users to specify service management requirements at a high level of abstraction in terms of different kinds of metadata: declarative management policies for migration, binding and access control, and profiles for the description of user preferences, device capabilities, and service component characteristics. Another key feature of CARMEN is the exploitation

of mobile middleware proxies that follow the provision-time movement of users, to customize service provisioning, to maintain service session state and to operate asynchronously with regards to temporarily disconnected clients. CARMEN implements mobile proxies in terms of Mobile Agents (MAs). With the adoption of MA-based proxies, wireless portable devices need limited network connectivity, only to inject in the fixed network the responsible proxies acting on their behalf. At the highest level of CARMEN architecture there are a Metadata Manager (MM) and a Context Manager (CM). The first supports the specification, modification, check for correctness, installation and evaluation of profiles and policies; the latter dynamically determines the context of a CARMEN client, supports the accessibility of resources and manages resource bindings in case of context modifications. The CARMEN low-level facilities provide mechanisms and tools to address most common issues in context-aware service provisioning to wireless clients.

With respect to CARMEN, NAM framework is more focused in service composition than in context-aware service provisioning; moreover CARMEN proxies do not take into account device capabilities and resource as NAM does, because they migrate to more powerful nodes of the network to exploit computation intensive operations.

2.1.4 NSAM for p2p service-oriented infrastructure

The NSAM model is particularly suitable to characterize service-oriented peers, interacting with complex environments (figure 2.3). In this framework, among the resources of the peer (R set, according to the NAM/NSAM model), functional modules are implemented as services. For example, the overlay scheme mechanisms are implemented as atomic services that each NSAM runs. Considering a group of NSAMs, the peer-to-peer interaction of their overlay services leads to the emergence of a composite overlay service. Self-configuration works similarly, with atomic services that adapt the configuration of the NSAM, based on information that in general is both local and external, for which a composite service spanning the whole network drives a global adaptation process. Thus, service composition mechanisms are embedded in the implementation of atomic services.

The CTRL component of the NSAM is basically a lightweight resource manager,

that configures the NSAM at startup, deciding which resources must be run, and manages their runtime allocation. In particular, it defines and implements the instantiation policy of atomic services that are requested by multiple consumers at the same time.

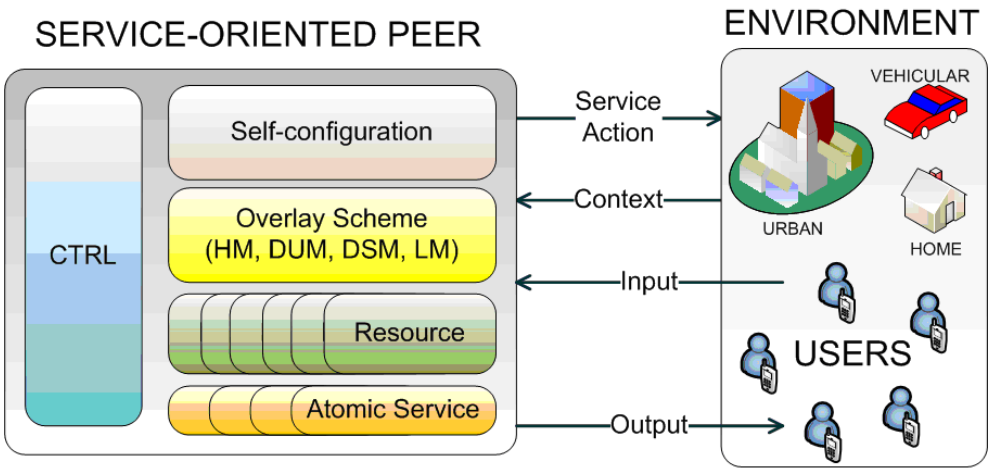

Figure 2.3: The structure of a service-oriented peer, supporting ubiquitous computing for mobile users in highly dynamic and heterogeneous environments.

Service composition is highly desirable in peer-to-peer (P2P) systems where application services are naturally dispersed on distributed peers. However, it is challenging to provide high quality and failure resilient service composition in P2P systems due to the decentralization requirement and dynamic peer arrivals/departures. Moreover, in pervasive computing environments peers are hosted on a number of devices with heterogeneous functionality sets. In the presence of such variety, it is desirable to dynamically combine available basic services (as building blocks) to create composite services.

Dynamic composition mechanisms built using graph techniques [33] provide support to user tasks in the face of dynamic challenges such as heterogeneity, resource restrictions, user and resource mobility, locality of service provisioning, and so forth. It is also necessary to dynamically capture information regarding the state

of a device while the device is operational. Such a dynamic mechanism will ensure uniform resource consumption, timely support, and fairness in resource utilization. A P2P service overlay network (figure 2.4) may be defined, over which service consumers send requests to service providers and new services can be flexibly composed from available service components based on the user's function and quality-of-service (QoS) requirements. However, in general, mobile devices still have difficulties in fully satisfying users' requirements, due to shortcomings in system resources, especially limited battery life. Restrictions in battery capacity prohibit the use of fully functional applications for satisfactory durations. In addition, the mobile computing environment requires applications to adapt dynamically to their context, including the user's role, capability, and current environment, while maintaining the constant functionality of applications. When an application invokes a complex task that can be performed by a combination of services, that application is resolved to a service composition or a service flow that is represented as a service composition graph. QoS control for applications running in a mobile peer must be invoked in a way that does not exhaust the resources of the device, including residual battery energy.

The metadata that represents the service includes descriptions about service capabilities. By employing semantics, formal declarative descriptions are attached to services. Semantic descriptions of services are used to organize services in a repository, retrieve the appropriate services and use them correctly. A domain ontology may used to conceptualize domain knowledge with commonly accepted vocabulary and to provide semantics to service descriptions. The syntactic parameters of a service define input, output, QoS parameters, pre-conditions and post-conditions, if present. Addressing the issue of mobility, within the P2P service overlay, it is possible to ensure that the request is recomputed with minimal interruprion of the session under progress. The effect of user mobility while a service session is in progress can lead to a complete dynemic recomposition of the service.

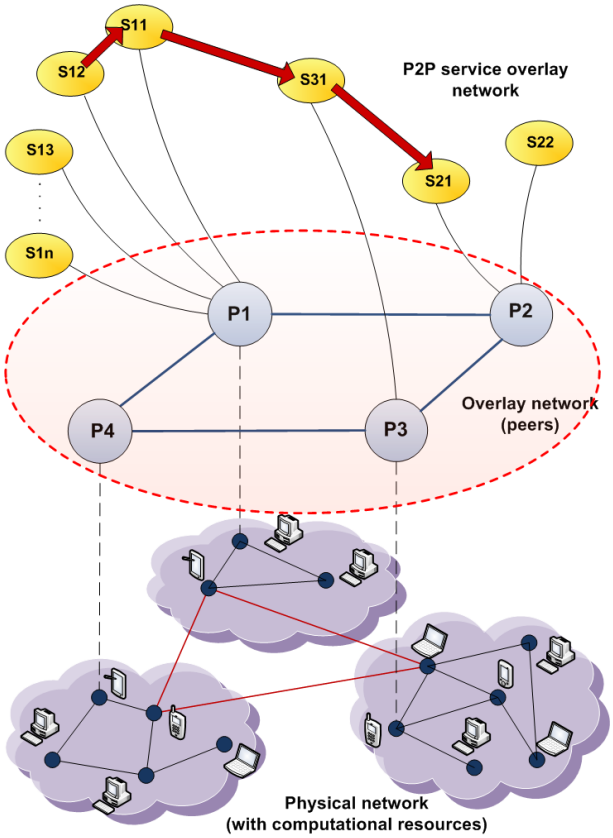

Figure 2.4: The three network layers: P2P service overlay network, peers' overlay network, and physical network.

2.2 Code Mobility

According to the definition given by Carzaniga *et al.*, code mobility is the capability to reconfigure dynamically, at runtime, the binding between the software components of the application and their physical location within a computer network [34]. The presented modeling formalism NAM is particularly suitable for building peer-to-pper systems characterized by services that migrate on-demand, thus allowing to cope with highly dynamic environmental conditions.

Mobile code technologies can be analyzed by considering the ability to transfer the state of an execution thread (or execution unit, to use a more general term). We say that a technology supports *strong mobility* if it allows executing units to move their code and execution state (*e.g.*, the stack and instruction pointer of a thread) to a different site. When an executing unit must be transferred to a remote site, it is suspended, transmitted to the destination site, and resumed there. A technology supports *weak mobility* if it allows an executing unit in a site to be bound dynamically to code coming from a different site (*i.e.*, the code can be moved and executed automatically), but no execution state is transferred across the network. This means that even though some data state could be transferred, the executing unit would need to be restarted upon arrival [35].

	Before		After	
Paradigm	Site A	Site B	Site A	Site B
Client Server	A	Know-how Resources B	A	Know-how Resources B
Remote Evaluation	A Know-how	B Resources	A	Know-how Resources B
Code On Demand	A Resources	B Know-how	Know-how Resources A	B
Mobile Agent	A Know-how	Resources	-	Know-how Resources A

Figure 2.5: MC taxonomy.

An acceptable classification of mobile code paradigms is illustrated in figure 2.5. It is based on the assumption that a component A, placed at site S_A, needs the results of the computation of a service. We assume the existence of another site S_B, involved in the delivery of the service. To obtain the service results, A starts the interaction pattern that leads to service delivery. Service execution involves a set of resources, the know-how about the service (its code), and a computational component responsible for the execution of the code [35]. Four solutions are possible: Client/Server (CS), Remote Evaluation (REV), Code on Demand (COD) and Mobile Agent (MA).

The first solution, CS, is static: components cannot change their location or their code during their lifetime. The REV and MA allows remote execution, respectively by means of code uploading or component migration. Finally, COD allows a computational components to retrieve code from another remote component. Within the context of the formal framework that specifies the concept of *Networked Autonomic Machine (NAM)*, resources provided by each NAM may be shared and also migrated within a group of NAMs.

2.2.1 Resource and Service migration

In the NAM framework, a resource migration is defined as a tuple

$$M =< NAM_u, NAM_d, R > \tag{2.8}$$

where R is the resource, NAM_u is the node providing it (uploader) and NAM_d is the node receiving it. If the resource is a service σ, before its migration can start it is necessary to match the resources of the downloader with the set R_σ of resources required by the service itself (*e.g.* running environment, libraries, CPU, RAM, etc.).

Chapter 3

Middleware and Applications

The definition and design of NAM framework has its basis in two projects that have been developed and managed by the Distributed System Group of University of Parma: *SP2A (Service-oriented Peer-to-peer architecture)*, a lightweight middleware enabling service-oriented peers for efficient and robust distributed applications, and *JXTA-SOAP*, a component which extends the JXTA middleware, with the design goal of wrapping Web Services in JXTA services through the use of JXTA for Web Service discovery and SOAP message transport. JXTA-SOAP has been included in SP2A middleware as an external API, thus enabling Web Services technology for the development of Service Oriented Peers (SOPs).

During the Ph.D. research activity several applications have been developed, targeting different fields, from Ambient Intelligence to e-learning communities, to emergency management scenarios, and finally to logistics, with the goal of enabling interoperability among heterogeneous platforms (including resource constrained devices) and ubiquitous sharing of services and resources. In the following, these applications, which have been implemented within SP2A and JXTA-SOAP, are presented, with emphasis on the enhancements that it is possible to obtain with the introduction of NAM framework.

3.1 Ubiquitous p2p sharing of services: JXTA-SOAP mobile

JXTA-SOAP component has been designed having in mind ubiquitous computing needs. JXTA-SOAP extends JXTA which is a set of open, generalized peer-to-peer protocols that allow a vast class of networked devices (smart phones, PDAs, PCs and servers) to communicate and collaborate seamlessly in a highly decentralized fashion. JXTA-SOAP aims at reducing the complexity otherwise required to build and deploy peer-to-peer service-oriented applications, by providing an open source software platform and deployed virtual network. The JXTA platform provides core building blocks (IDs, advertisements, peergroups, pipes) and a default set of core pluggable policies. JXTA-SOAP completes this framework by providing service-related mechanisms which have been designed taking into account many important software design patterns. JXTA-SOAP is currently implemented in two versions: J2SE-based (fully featured, extending JXTA-J2SE) and J2ME-based (partially featured, extending JXTA-J2ME). The package structure is the same for both J2SE and J2Me version (figure 3.1). External libraries (`libs`) and example(`examples`) folders are in the main path of the project; on the contrary there are distinct paths for the two implementation of the component, namely `soap.j2se` and `soap.cdc`.

The J2SE version of the JXTA-SOAP component supports both service deployment and service discovery/invocation. It is based on Apache Axis 1.4, which is an implementation of the SOAP protocol. In typical client/server settings, Axis is deployed in a servlet engine, such as Apache Tomcat, along with implemented Web Services, while client applications use Axis' Java API to create instances of requests. The so-called Axis engine is the running processing logic, either client or server. The J2ME version of the architecture supports Connected Device Configuration (CDC) and Personal Profile. We implemented the API which enables the development of peers that are able to deploy, provide, discover and consume Web Services in a JXTA-SOAP network. Since Axis is not available for the CDC platform, we adopted kSoap2 [36] as SOAP parser (for consumer functionalities) and, for service provision, we integrated the mAS [22] lightweight engine.

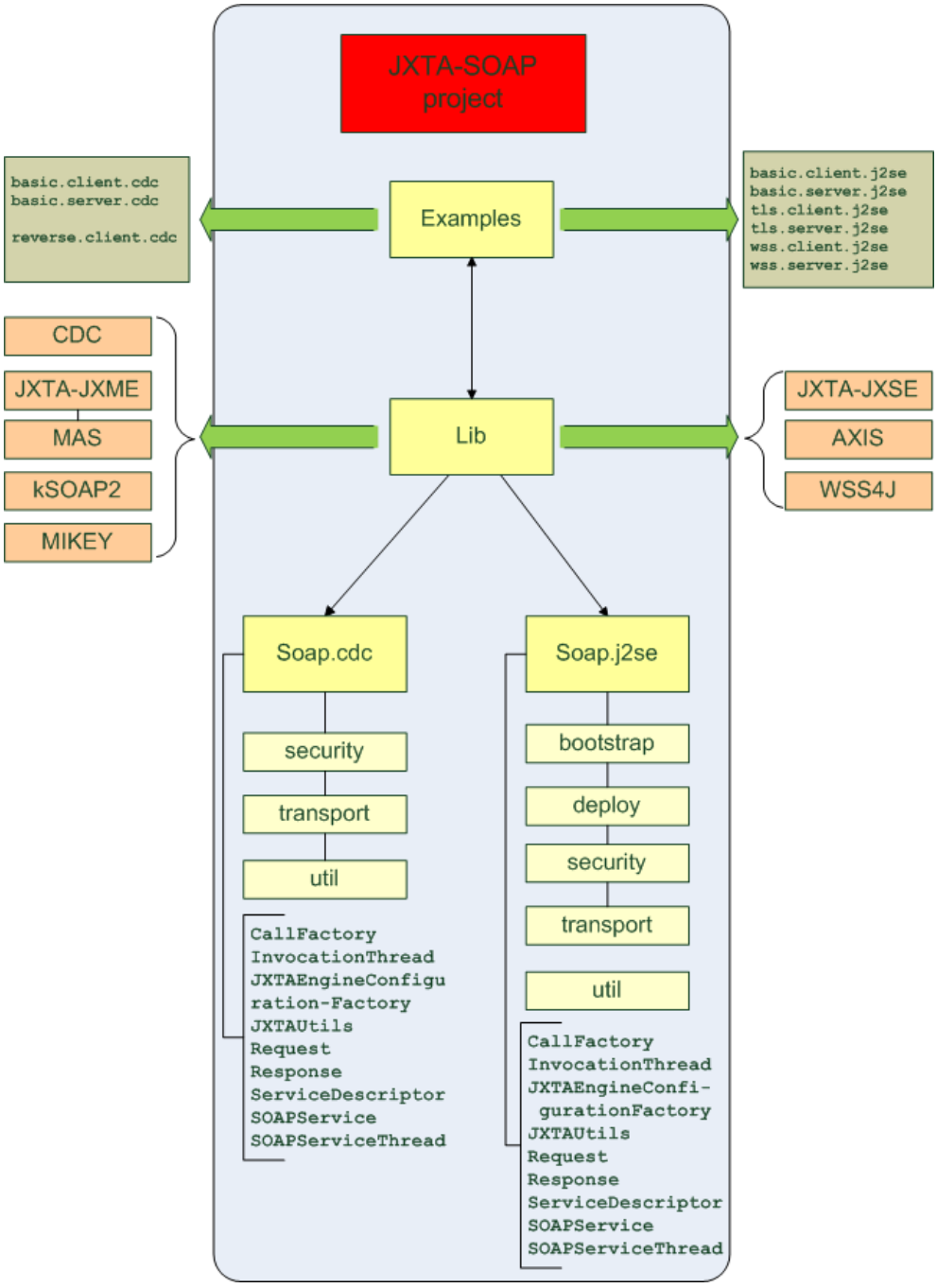

Figure 3.1: Package structure of Jxta-Soap component

In the following the internal architecture of JXTA-SOAP is described, with reference to the features and the different technological solutions the J2ME version of the component relies on.

3.1.1 Service deployment

In order to deploy its services, a JXTA-SOAP based peer has to instantiate and configure the related SOAPService objects (one for each hosted service), and to advertise the service interfaces in the network. The class diagram in figure 3.2 illustrates the relationships among classes which are involved in this tasks. The Peer class represents the generic peer application implemented by the developer, and relies on JXTA-SOAP's API which provides all the other classes represented in the diagram.

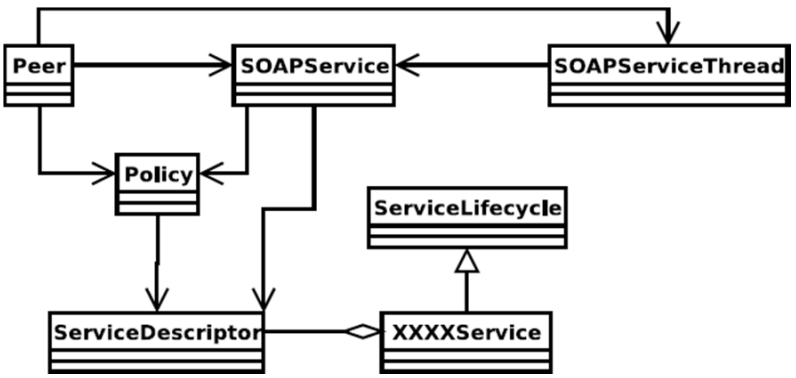

Figure 3.2: Classes involved in service deployment.

At start-up, the typical Peer object bootstraps the JXTA-JXME platform, configuring basic connectivity settings, such as TCP/IP and HTTP ports, and joins the public JXTA peergroup (represented by an instance of PeerGroup class). Bridges to existing common membership and access technologies are the MembershipServices, which allow peers to establish their identity within peergroups. Each MembershipService implementation is responsible for its own protocol definition. The default membership service for the public JXTA peergroup is the PSEMembershipService, based on *Personal Security Environment (PSE)*. This service initializes the PSE KeyStore,

an object that acts as a secure datastore for peer certificates and keys. Once the Peer has authenticated itself to PSE, it is ready to deploy its service instances.

For each service to be deployed, a ServiceDescriptor must be instantiated. The ServiceDescriptor is filled with service-specific information, such as the service name, a brief description, the implementation class name, the peergroup ID, the security tag. Moreover, the ServiceDescriptor implements the ServiceLifecycle interface, which is used to pass a *Context Object* [37] (constructed by the user application) to the Web Service class.

Next step is to create the context object, and to pass it to a SOAPService instance, along with a security Policy that is associated to each service implementation.

The SOAPService class is neuralgic, since it provides all basic service-related operations. The SOAPService's initialization method sets the ServiceDescriptor, bootstraps the service provision engine, creates and publishes the public pipe and the advertisement of the service. Since the J2ME version of JXTA-SOAP does not support Axis, we have integrated the Server class of the Micro Application Server (mAS) into the basic service class of the JXTA-SOAP API. mAS implements the Chain of Responsibility pattern [38], the same used in Axis. It avoids coupling the sender of a request to its receiver by giving more than one object a chance to handle the request; receiving objects are chained and the request passed along the chain until an object handles it. Moreover, mAS allows service invocation by users and service deployment by the owner, and supports browser management of requests, distinguishing whether the HTTP message contains a Web page request or a SOAP envelope. Like it happens with Axis, when mAS runs, a series of Handlers are each invoked in order. The object which is passed to each Handler invocation is a MessageContext. A MessageContext is a structure which contains several important parts: 1) a "request" message, 2) a "response" message, and 3) a bag of properties.

For each service implementation, the Peer spawns a SOAPServiceThread which waits for other peers' connections to that service, on the service public pipe, and creates a pool of InvocationThreads to efficiently execute service invocation (the *Thread-pool* pattern is illustrated in [39]).

3.1.2 Service publication and lookup

The main enhancement of JXTA-SOAP with respect to traditional Web Service frameworks is the adopted distributed approach for service advertising and lookup. JXTA-SOAP allows to encapsulate WSDL interfaces in particular JXTA documents, the so-called ModuleSpecAdvertisements, which can be spread into the network using many different routing policies which can be inserted in JXTA protocol stack. In general, a ModuleSpecAdvertisement describes a module specification. Its main purpose is to provide references to the documentation needed in order to create conforming implementations of that specification. A secondary use is to make running instances remotely usable, *e.g.* by publishing a pipe advertisement associated to the ModuleSpecAdvertisement. Once the ModuleSpecAdvertisement has been filled with the information about service (WSDL for J2SE version and service main class and methods for J2ME version) and the secure pipe tag, a context object is created and passed to the SOAPService instance. Moreover, the Peer spawns a SOAPServiceThread which waits for other peers' connections to the deployed service, on the public pipe associated to the ModuleSpecAdvertisement.

Service publication is a distributed process, which uses network nodes as a distributed repository (on the contrary, traditional UDDI registries are centralized interface description repositories). As for publication, service lookup is a distributed process, which can be conceptualized as message exchange between low-level JXTA modules.

3.1.3 Service invocation

SOAP is the most widely used Web Service protocol for message enveloping. SOAP message management in JXTA-SOAP J2ME version is based on kSoap2, while the standard version is based on Axis. Figure 3.3 illustrates associations among classes which are involved in the service invocation task, performed by a generic Peer. The latter, once it has discovered the ModuleSpecAdvertisement and the interface of the required Web Service, creates a SOAPTransportDeployer, which manages the transmission of SOAP messages to and from the service using its pipe. Moreover, the

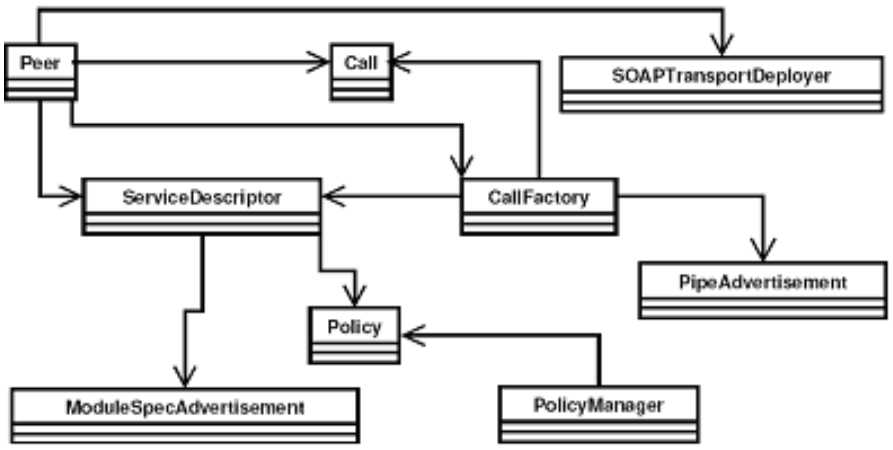

Figure 3.3: Classes involved in service invocation.

Peer creates a ServiceDescriptor, which is used by the kSoap2 based implementation CallFactory.

The latter instantiates a kSoap2's Soap Object, and sets all the properties for message exchanging through JXTA pipes. Soap Object is a highly generic class which allows to build SOAP calls, by setting up a SOAP envelope. We have maintained the same structure of J2SE-based version for Call Factory, to allow portability of service consumer applications from desktop PCs or laptops to PDAs. Internally, the Call Factory class creates a Soap Object passing references to the Service Descriptor, the public pipe advertisement of the service and the peergroup as parameters for the creation of the Call object.

The Call Factory class also allows to create an instance of kSoap Pipe Transport, the class we implemented to manage the transmission of SOAP messages using service pipes. The kSoap2 API provides a Transport class that encapsulates the serialization and deserialization of SOAP messages, but does not manage communication with the service; the HTTP Transport subclass, both in CDC and CLDC version, allows service invocation over HTTP, setting up the required properties, but it uses URLs as absolute references of remote services, and it is not suitable for usage

in JXTA-SOAP, where services (as every resource) are identified by JXTA-IDs and must be invoked through JXTA pipes. Thus, we extended the Transport class with the implementation of a call functionality that configures a JXTA pipe and creates the messages to be sent over it.

After instantiating the transport using the Call Factory class, the consumer peer creates the request object, indicating the name of the remote method to invoke and setting the input parameters as additional properties. This object is assigned to a Soap Serialization Envelope, as the outbound message for the soap call; Soap Serialization Envelope is a kSoap2 class that extends the basic Soap Envelope, providing support for the SOAP Serialization format specification and simple object serialization. The same class provides a getResponse method that extracts the parsed response from the wrapper object and returns it.

3.1.4 Secure service invocation in JXTA-SOAP

Peer-to-peer architectures present a particular challenge for providing high levels of availability, privacy, confidentiality, integrity, and authenticity, due to their open and autonomous nature. Network nodes cannot be considered trusted parties, and no assumptions can be made regarding their behavior. Preserving integrity and authenticity of resources means safeguarding the accuracy and completeness of data and processing methods. Unauthorized entities cannot change data; adversaries cannot substitute a forged document for a requested one. Privacy and confidentiality mean ensuring that data is accessible only to those authorized to have access, and that there is control over what data is collected, how it is used, and how it is maintained. A malicious node might give erroneous responses to requests, both at the application level, returning false data, or at the network level, returning false routes and partitioning the network. Moreover, the P2P system must be robust against a conspiracy of a malicious collective, i.e. a group of nodes acting in concert to attack reliable ones. Attackers may have a number of goals, including traffic analysis against systems that try to provide anonymous communication, and censorship against systems that try to provide high availability.

Security attacks in P2P systems can be classified into two broad categories: pas-

sive and active [40]. Passive attacks are those in which the attacker just monitors activity and maintains an inert state. The most significant passive attacks are *eavesdropping*, which involves capturing and storing all traffic between some set of peers searching for some sensitive information (such as personal data or passwords), and *traffic analysis*, where the attacker not only captures data but tries to obtain more information by analyzing its behavior and looking for patterns, even when its content remains unknown. In active attacks, communications are disrupted by the deletion, modification or insertion of data. The most common attacks of this kind are: *spoofing*, in which one peer impersonates another; *man-in-the-middle*, where the attacker intercepts communications between two parties, relaying messages in such a manner that both of them still believe they are directly communicating; *playback* or *replay*, in which some data exchange between two legitimate peers is intercepted by the attacker in order to reuse the exact data at a later time and make it look like a real exchange; *local data alteration*, which goes beyond the assumption that attacks may only come from the network and supposes that the attacker has local access to the peer, where he can try to modify the local data in order to subvert it in some malicious way.

To cope with malicious attacks, security policies adopted at the overlay P2P network level usually consist of key management, authentication, admission control, and authorization. These are the strategies we took into account for securing consumer-to-service communication in JXTA-SOAP. Currently, JXTA-SOAP supports secure service invocation by means of two orthogonal mechanisms. The first one, *transport-level security*, allows to create a secure channel which guarantees the integrity and confidentiality of exchanged information, by means of mutual authentication between parties (using certificates) and data encoding. The other approach is WSS-based *message-level security*, for which SOAP messages sent by service consumers contain security parameters (tokens) which are extracted by service providers to check for consumers' compliance with the security policy of the invoked service. Figure 3.4 summarizes the interaction of a peer with a discovered secure service.

The `net.soap.jxta.security.policy` package provides two important modules, *i.e.* `Policy` and `PolicyManager`. The latter is a class which allows to associate a service with a security policy (currently: TLS, WSS, or both). The

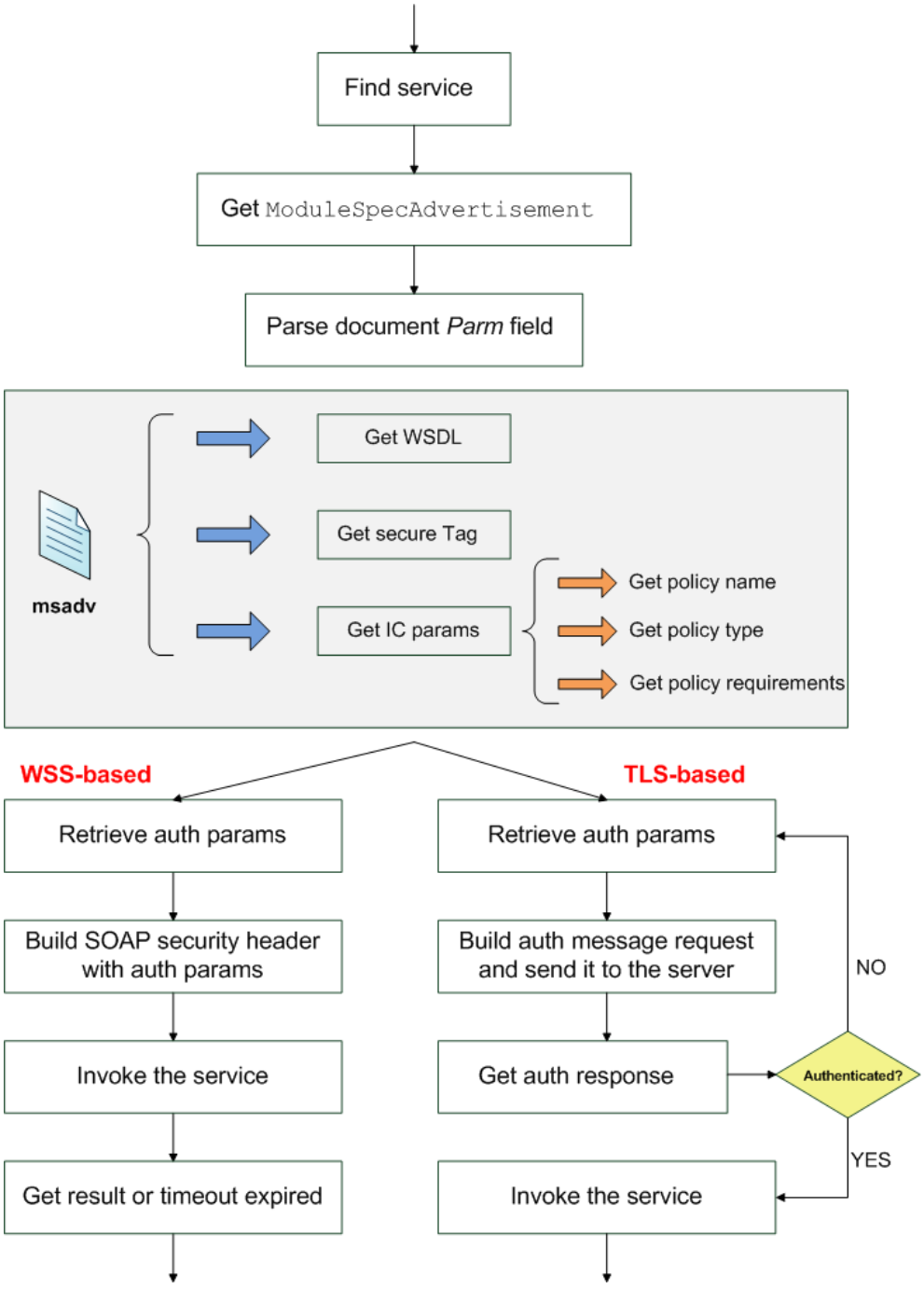

Figure 3.4: Interaction with a discovered secure service.

`Policy` interface provides methods which are commonly implemented by all policy classes. They allow to extract authentication parameters from the client request message and to verify their correctness according with the service security requirements. If all parameters are validated, the client is allowed to invoke the service, otherwise the request is refused. Authentication parameters are stored in a vector of `SOAPService` class that is used to keep a list of authenticated peers.

Current implementations of the `Policy` interface are `DefaultTLSPolicy`, based on `JXTAUnicastSecure` pipes which subsume default (unsecure) JXTAUnicast pipes, and `DefaultWSSPolicy`, which uses Apache's WSS4J to provide SOAP messages with security headers and fill them with tokens. Policies are associated to services by means of two extensions of the `<Parm>` field of the `ModuleSpecAdvertisement`. The first extension is the `<security>` tag, whose value (true or false) indicates whether a service is secured or not. The other extension is the `<InvocationCharter>`, a XML document which is inserted in the `<Parm>` field of the ModuleSpecAdvertisement if the `<security>` tag is set to true. The secure invocation model is illustrated in figure 3.5 (TLS-based case), with particular emphasis on multithreaded handling of concurrent invocations.

The `net.soap.jxta.security.certificate` package provides classes that enable JXTA-SOAP to use X.509 certificates for authentication and to extract them from a PeerAdvertisement; in particular, it uses the Key Store associated to JXTA PSEMembershipService, that is initialized when the peer boots in the network group. The `net.soap.jxta.security.wss4j` package implements a WSSecurity class, that enables the client to create a custom security header for invocation requests, according to the Invocation Charter (*i.e.* encryption, body signature).

DefaultTLSTransport

TLS is the default technology used by JXTA-SOAP when the secure service is created. Each SOAP message between client and server is sent through a TLS channel which requires previous authentication of involved entities. According to the implemented policy, a communication session is established, in which the client sends to the server its PeerAdvertisement with the peer self-signed certificate created by

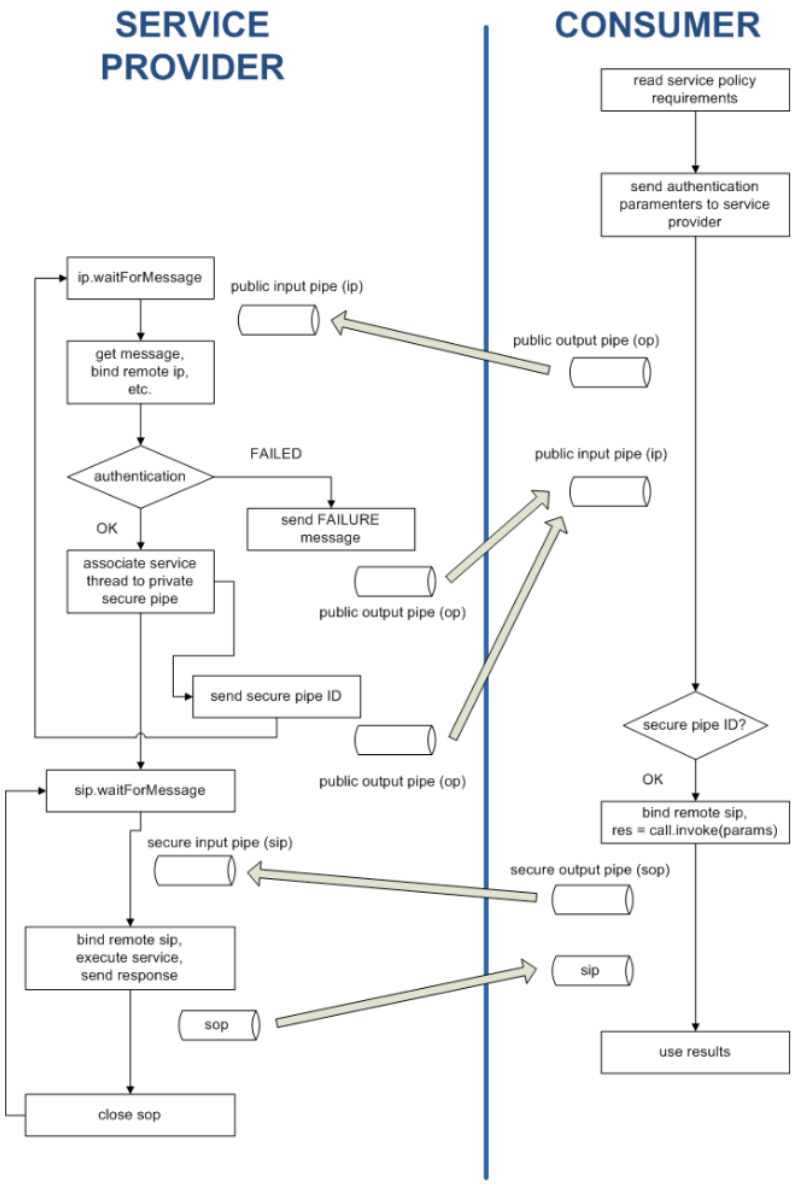

Figure 3.5: TLS-based secure invocation model. Concurrent invocations are handled by separated service threads.

JXTA. The server is able to extract additional information about the requesting client (i.e. name, PeerID, group) and, if the authentication procedure succeeds, to update the list of authenticated peers with the corresponding entry:

[PeerID, X.509 Certificate]

Then a TTL (time to live) is associated to the authentication of the client. At the end of the period of validity, the entry is automatically deleted from the list.

In JXTA, the default Membership Service is PSE, which stands for Personal Security Environment. This service is the only one that is considered secure. PSE provides credentials based on X.509 certificates. Any number of such certificates may be included as Certificate elements in the PSE credential, together with the Peer Group ID and the subject's Peer ID. The credential itself is also signed. Since the default TLS policy implementation uses JXTA's PSE keystore for storing/retrieving certificates, it is necessary that both client and server also authenticate to the PSEMembershipService, which is the default membership service implemented in JXTA. When the client request is received by the server, an output pipe is created for invocation response and validation of authentication parameters; then the X.509 certificate is imported, a new thread is created for secure invocation management and the associated secure pipe is sent to the client for successive invocations.

DefaultWSSMessage

This is the default message security policy and uses the methods of WSSecurity class to build a security header which is included in all messages and invocation requests. During the authentication phase, the client attaches its X.509 certificate to the header using a `<wsse:BinarySecurityToken>` and digitally signs the request message body with its private key.

The WSS-based policy does not use JXTA PSE keystore, but requires that both client and server generate a couple of private/public keys and use them to create a self-signed X.509 certificate which is stored in their own keystore, whose integrity and privacy are granted by means of a password.

MIKEYPolicy for mobile applications

Since PSE membership classes are not available for JXME version of JXTA platform, we imported and modified them to be able to authenticate peers within a peer group. Moreover, J2ME does not support TLS, so it was impossible to use JXTA secure pipes for service invocation. We implemented a new type of pipe, `JxtaUnicastCrypto`, by which it is possible to cipher message contents, and we used them to define a new security policy, suitable for Connected Device Configuration (CDC) and Personal Profile.

`PipeService` and `PipeServiceImpl` classes in the `net.jxta.pipe` package create and use a SecretKey object containing the cipher algorithm and the corresponding key. The client and the server have to share the same key, so we introduced Multimedia Internet KEYing (MIKEY) [41] protocol to create the key pair and all the required parameters for encryption and decryption operations. Although memory and processing power have dramatically improved for handheld devices, encryption remains a resource-intensive task that requires consideration when designing protocols. MIKEY is a schema for management of cryptographic keys which can be used in real-time and peer to peer applications; it was developed with the intention to minimize latency when exchanging cryptographic keys between small interactive groups that reside in heterogeneous networks. The protocol is defined in RFC 3830 and in JXTA-SOAP project we introduced an implementation with RSA-R algorithm [42]. The standard describes mechanisms for negotiating keys between two or more parties who want to establish a secure channel of communication; to transport and exchange keying material, three methods are supported, Pre-shared secret key (PSK), Public Key encryption (PKE) and Diffie Hellmann (DH) key exchange. Each key-exchange mechanism (PSK, PKE, and Diffie-Hellman) defined in MIKEY is using the same approach of sending and receiving messages, but the message attributes (that is, headers, payloads, and values) differ from method to method.

To create a MIKEY message it is necessary to create an initial MIKEY message starting with the Common Header payload, and then concatenate necessary payloads of the message. As a last step create and concatenate the MAC/signature payload without the MAC/signature field filled in; calculate the MAC/signature over the en-

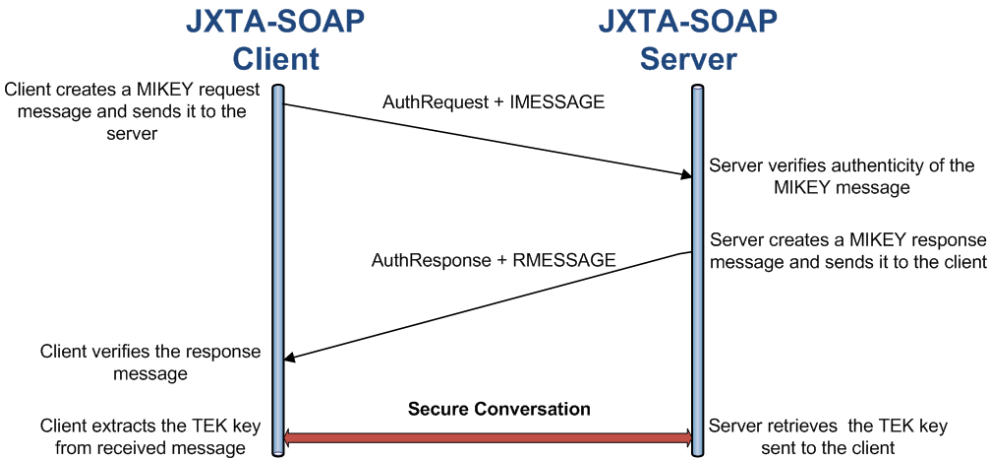

Figure 3.6: MIKEY transaction example: the JXTA-SOAP Client is a peer that acts as service consumer, while the JXTA-SOAP Server is a peer that acts as service provider. Of course roles can be exchanged, since every peer can provide and consume services.

tire MIKEY message, except the MAC/Signature field, and add the MAC/signature in the field. The common header payload must be included at the beginning of each MIKEY message (request and response) because it provides necessary information about the Crypto Session with which it is associated. MIKEY defines several payloads to support the three key exchange methods and the corresponding architectural scenarios (that is, peer to peer, simple one to many, many to many, without a centralized control unit). The main characteristic of MIKEY protocol is that it minimizes message exchange; the negotiation of key material should be accomplished in one round trip, as described in figure 3.6.

Ubiquitous computing, for its nature, has an extremely wide range of applications. Here we consider two important and challenging fields, *i.e.* ambient intelligence and emergency management, for which we have developed solutions based on JXTA-SOAP.

3.1.5 Ambient Intelligence applications

Ambient Intelligence (AmI) refers to digital environments that proactively support people in their daily lives, based on the convergence of three key technologies: Pervasive Computing, Artificial Intelligence, and Intelligent User Friendly Interfaces [**?**]. AmI represents a step beyond the current concept of User Friendly Information Society, because the technologies should be fully adapted to human needs and cognition. Indeed, AmI should be orientated towards community and cultural enhancement, helping citizens to build knowledge and skills, and to achieve better quality of life. At the same time, AmI should inspire trust and confidence, working in a seamless, unobtrusive and often invisible way.

In the AmI context, the European Commission recently started the Ambient Assisted Living (AAL) technology and innovation funding programme, aiming at extending the time older people can live in their home environment by increasing their autonomy and assisting them in carrying out activities of daily living, feeling included, secure, protected and supported. AAL spaces are physical places featured with AmI enabling technologies, including the intelligence which supports the services. Examples of AAL spaces are the home where the user lives, the neighborhood, the town, but also the body of the user itself. The technical challenge is to develop an integrated technological platform that allows the practical implementation of the AAL concept for the seamless and natural access to those services indicated above, to empower the citizen to adopt ambient intelligence as a natural environment in which to live. In particular, the novel paradigms Adaptive Services/Client Paradigm (SCP) and Spontaneous Emergence Paradigm (SEP) described in chapter 1, are particularly suitable for regulating the interactions among the software entities of AAL-oriented AmI systems.

User Activity Monitoring

One of the most challenging AmI services is User Activity Monitoring, which may be transversal to every AmI scenario. The framework illustrated in chapter 2, in conjunction with the SCP and SEP paradigms, is able to provide the flexibility required

to deal with highly dynamic environments where devices continuously change their availability and (or) physical location (e.g. those which are carried or worn by the user). This complex problem of composing and decomposing connections among nodes is abstracted in an overlay network where the Activity Monitor (AM) component subscribes for raw context events coming from other distributed components (sensors, specialized data filters, etc.), searches for remote services which may provide useful information for its reasoning function, and publishes context events which describe indoor and outdoor activity of the user, taking into account different contour information such as medical prescriptions, planned agenda, etc.

A distinction between static and dynamic activities is necessary. Static activities like "standing" or "sitting" can be inferred directly from the low-level data at a particular time instant (such as the pose of the person at a certain time using some kind of thresholding mechanism on the pose estimate). By contrast dynamic activities, such as "moving around", are usually composite activities requiring a monitoring of a full sequence of low-level data (e.g. context events describing ongoing sub-activities). Low-level data needs to be stored for several time frames (in a context buffer), as the whole sequence is needed to infer that activity from an evolution of the low level data. For example: "cooking" may be composed of several low-level data at different time instances: "opening the fridge", "closing the fridge", "standing in front of the oven", etc. Outdoor user activities are even more challenging to detect. The user may wear a personal mobile device (PMD) and sensors that monitor the level of its activity. The PMD should have a mechanism to be called from an external entity to deliver the activity level. Thus, the mobile device would be both service provider and service consumer. Collected information, which is analyzed in deferred time, may be useful for several other AAL services, *e.g.* planning the weekly menu (the less activity, the less amount of calories to ingest).

3.1.6 Emergency Management application

Emergency management (or disaster management) is the discipline of dealing with and avoiding risks [43]. It involves preparing for disaster before it happens, disaster response (*e.g.* emergency evacuation, quarantine, mass decontamination, etc.), as

well as supporting and rebuilding society after natural or human-made disasters have occurred. The disaster management cycle involves four key phases:

1. Mitigation: includes any activities that prevent a disaster, reduce the chance of a disaster happening, or reduce the damaging effects of unavoidable disasters.

2. Preparedness: includes plans or preparations made to save lives or property, and to help the response and rescue service operations.

3. Response: includes actions taken to save lives and prevent property damage, and to preserve the environment during emergencies or disasters. The response phase is the implementation of action plans.

4. Recovery: includes actions that assist a community to return to a sense of normalcy after a disaster.

These four phases usually overlap. Information and Communication Technology (ICT) is being used in all the phases, but the usage is more apparent in some phases than in the others. For example, ICT support is very important during the disaster response (DR) phase of an emergency, which may commence with search and rescue, but in all cases the focus will quickly turn to fulfilling the basic humanitarian needs of the affected population. This assistance may be provided by national or international agencies and organizations. Effective coordination of disaster assistance is often crucial, particularly when many organizations respond and local emergency management agency capacity has been exceeded by the demand or diminished by the disaster itself. Tracing missing people, coordinating donor groups, recording the locations of temporary camps and shelters are examples of problems in the immediate post-disaster period that can be effectively addressed by using ICT.

Disasters can happen anywhere at any time. Some disasters can be prevented, while some others cannot. Preparedness however greatly increases our chances to reduce their impact. Developing effective early warning and alert systems often can save thousands of human lives. From the 2004 tsunami in the Indian Ocean to the forest fires that ravaged southern Europe in the summer of 2007, recent natural and

man-made disasters (including also conflict-related complex emergencies) have highlighted the need for a more effective response.

In the area of civil protection the European Commission has recently proposed to improve the EU's capacity through a number of important measures [44]. Among others, building up the Monitoring and Information Centre (MIC), playing the role of operational center for European civil protection intervention. This requires a qualitative shift from information sharing/reacting to emergencies towards proactive anticipation/real time monitoring of emergencies and operational engagement/coordination. This includes early warning systems, performing needs assessments, identifying matching resources, and providing technical advice on response resources to the Member States; developing scenarios, standard operating procedures and lessons learned assessments; implementing the Commission competencies to pool available transport and provide co-financing for transport; increasing training and exercise activities for Member States and other experts; and helping the Member States to set up common resources. This implies also the use of monitoring capabilities such as those developed under the Global Monitoring for Environment and Security (GMES) initiative [45] or enabling tools like GALILEO (the European satellite navigation system) [46].

We have focused on disaster response exploitation based on the concept of *ubiquitous computing*, whose main objective is to provide globally available services and resources in a network by giving users the ability to access them anytime and anywhere.

Proposed technological framework

Our framework focuses on the problem of *identifying matching resources* in response to disasters. The most important are human resources, *i.e.* Civil Protection volunteers, Red Cross doctors and medical attendants, firemen, policemen, army officers, etc. In a typical scenario, it is necessary to coordinate the action of rescuers that are already in the disaster location, and those that are on vehicles and may be requested to reach the disaster place. The purpose of our work is also to support the work of the back-end operators, improving the ICT infrastructure that must allow not only communications among actors, but also automated gathering, elaboration and delivery of the huge

amount of data collected by each actor.

For example, in case of flooding, first volunteers arriving at the disaster location may notice that some roads are interrupted. If they are equipped with a mobile device including a camera, they may (1) send short alert messages, including their coordinates obtained by means of GPS/GIS, and (2) take and send photos to provide a more detailed description of the environment. The back-end system collects and filter these data, and sends useful advices (such as the best route to be followed) to rescue vehicles which are directed to the disaster place.

The infrastructure of the service-oriented applications we envision is a peer-to-peer overlay network, which is placed at level 5 in the TCP/IP stack and is almost independent from the possible connectivity solutions, that we summarize in the following.

For long distance communications, in Europe the most used infrastructure is General Packet Radio Service (GPRS), which is a packet-oriented Mobile Data Service available to users of Global System for Mobile Communications (GSM) and IS-136 mobile phones (the so-called second generation - 2G). It provides data rates from 56 up to 114 kbit/s. A more powerful technology which is assuming higher importance is the Universal Mobile Telecommunications System (UMTS), one of the third-generation (3G) cell phone technologies, which is also being developed into a 4G technology. Both 2G and 3G technologies require the presence of base stations on the territory. In case of heavy disasters such as hurricanes, base stations may be damaged, for which satellite and/or TETRA-based communications are the other options. TETRA is a telecommunications standard for Private Mobile Radio (PMR) systems developed by ETSI as an answer, at European level, to the evolving needs of PMR Operators, which have to cope with traffic congestion and a growing demand for speech and data services.

For local communications among actors equipped with mobile devices, infrastructured communications are usually based on WiFi. If some devices are out of the range of the WiFi access point, they can try to set up a mobile ad-hoc network (MANET), which is a self-configuring network of mobile routers (and associated hosts) connected by wireless links, the union of which form an arbitrary topology.

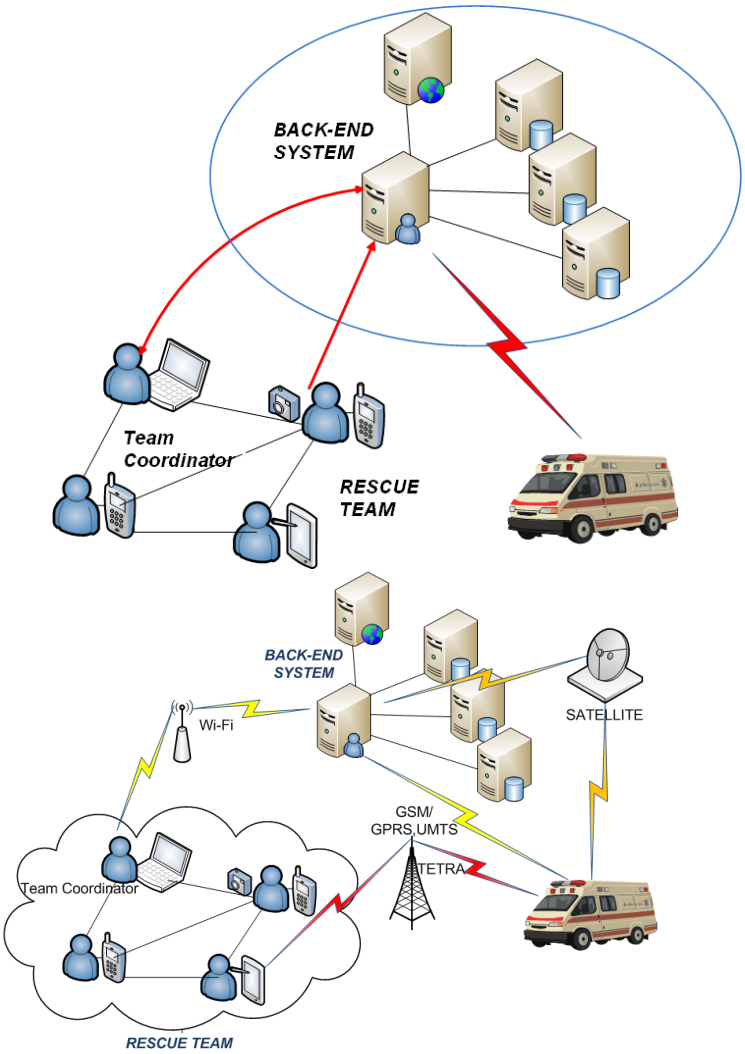

Figure 3.7: The back-end system, the rescue operators and vehicles are connected in a peer-to-peer overlay network, offering services to each others (left image). Connectivity is guaranteed by different technological solutions (right image).

The routers are free to move randomly and organize themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a standalone fashion, or may be connected to the larger Internet.

Example DR Application

Using JXTA-SOAP mobile, we developed a GUI-based application that allows to join a JXTA-based P2P network to share services for supporting disaster response activities. The application has several overlapping panels (or tabs), each one being related to a specific function. As illustrated in figure 3.8, the Remote panel shows discovered remote services. It is possible to search for services in the P2P network (offered by other rescue operators), and to select one of them from the resulting list, in order to see all the operations it offers, which are shown in the Operation tab. The user puts a description of the desired service in the search field, and all the matching services are listed in the table. Some services from the back-end are assumed to be always available, such as the one that provides photos taken by a satellite. The Operation management panel (figure 3.9) shows all the functionalities provided by the selected service; the operator can choose a particular operation and fill the input parameters table in the invocation panel.

3.1.7 Service-oriented Peer-to-peer architecture

The Service-oriented Peer-to-Peer Architecture (SP2A) [47] is a lightweight framework based on the Peer pattern [48], which defines the basic modules for building service-oriented peers (SOPs) for efficient and robust Grid environments. The SP2A middleware maps NSAM concepts in a thin software layer, independent from the hardware, whose resources are exposed as services. SP2A allows to cope with the requirements of applications with a large number of users dynamically connecting to the system, and provides high levels of scalability, decentralization and interoperability. It is distributed as a set of Java interfaces and both J2SE and J2ME class implementations, which support state-of-art technologies for peer-to-peer message routing, service description and deployment: Web Services, OWL-S and JXTA. These tech-

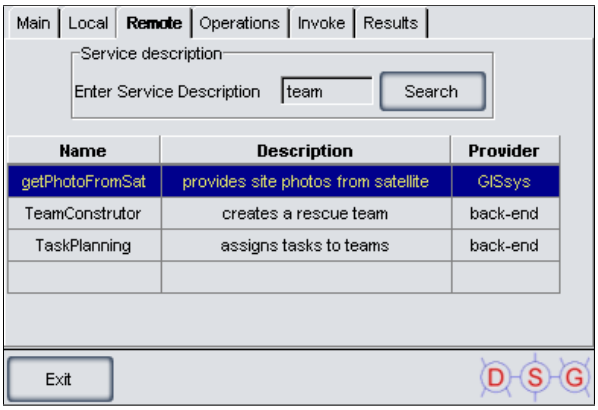

Figure 3.8: Disaster Response GUI: remote service selection panel

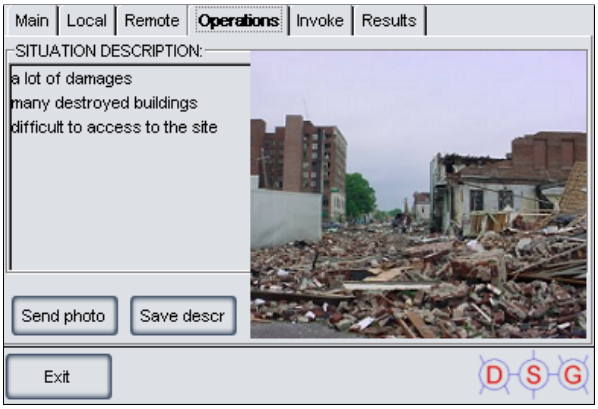

Figure 3.9: Disaster response GUI: operation management panel. A photo of the disaster location is taken, and a short description written, both ready to be sent to the back-end upon request, or proactively by the rescue operator.

nologies complement each others: Web Services provide a framework for service description; OWL-S supplies service providers with a core set of markup language constructs for describing the properties and capabilities of their services in unambiguous, computer-interpretable form; JXTA operates at the lower level providing P2P functionalities. According with the Peer pattern, the basic modules for building service-oriented peers are:

- *Communication Service(CS)*: allows peers to create a virtual network overlay on top of physical network infrastructure.
- *Resource Provision Services(RPSs)*: implement resource management mechanisms for local and remote control of Peer's resources.
- *Resource Monitoring Service(RMS)*: is responsible for providing a list of available local resources, maintaining related information, identifying and reporting failures.
- *Access Service(AS)*: is the entry point for users to interact with the system
- *Routing Service(RS)*: defines the rules for addressing, filtering sending and receiving messages
- *Scheduling Service(SCS)*: is responsible for managing task execution requests, based on information provided by the RMS
- *Security Service(SES)*: is responsible for protecting shared resources; it relies on mechanisms to safeguard integrity and authenticity of data, to ensure privacy and confidentiality in communications, and to provide means for user authentication and authorization. Moreover, the Security Manager allows peers to establish their identity within a group.
- *State Management Service(SMS)*: provides facilities to check and change the peer state.

Peergroup management

Peergroups are communities of peers organized for specific knowledge sharing. At the middleware level, the creation of subspaces is also motivated by the need to create scoping environments which restrict the propagation of query messages, thus improving the performance of discovery algorithms. Moreover, content exchange and service interaction often require the creation of secure domains. Peers can have different ranks, corresponding to the actions they are allowed to perform within the group. A partial list of ranks is:

- *admin* - the peer is a member trusted by the group founder; the actions it is allowed to perform are: service sharing/discovery, group monitoring, voting for changing member ranks;
- *newbie* - the peer is a new member; it only can search for an admin peer, to ask for a promotion;
- *searcher* - the peer is allowed to search for services and to interact with them;
- *publisher* - the peer can search for services but also publish its own services in the peergroup.

Service description, sharing and discovery

Resources are shared as a Resource Provision Services (RPSs); to use a resource, a consumer must know if the related Resource Provision Service (RPS) exists and is available; if it operates under a specified set of assumptions, constraints, policies; and if it can be invoked through a specified means, including inputs that the service requires and outputs that will form the response to the invocation. All SP2A RPSs have a name, a short textual description, and an uniquely identified owner. In addition, they expose an interface which specifies how to access its functionalities. Service deployment is transparent to the user, which only has to invoke the shareRPS method of the RPSManager. This encapsulates the creation of a JXTA service advertisement, and the creation of a service instance in a new thread.

```
// [1] Service construction
   MathService math = new MathService();
         rpsManager.addToRPSList(math);
// [2] Service activation and publication
   rpsManager.shareRPS(math);
```

SP2A allows to share in the P2P network both the descriptions of of "local" RPS, *i.e.* a service whose instance is intended to be running in separate thread of the SP2A-based peer application, as in the example above, and "external" RPSs, *i.e.*services deployed in traditional containers (*e.g.* Axis servers) and addressed by simple URLs.

Using the math example, we illustrate how a SP2A-based peer performs attribute/value search and service interaction, using the RPSManager and the methods which all RPSs have in common.

```
rpsManager.findRPS(mainGroup, "Name",
"MathService");
... // search results filtering
ResourceProvisionServiceImpl math =
(ResourceProvisionServiceImpl) rpsManager.
getDiscoveredRPSVector().getElementAt(..);
String WSDLbuffer = math.getInterfaceDescription("WSDL");
... // interface parsing
Integer a = new Integer(..);
math.invokeOperation("add", new Object[] { a });
```

Service selection and delivery

Service selection and delivery involve shared Resource Provision Services and other SOP modules. The Communication Service delivers a message to the Security Management Service, communicating resource request and credentials of the requester (another Peer); if the requester's credentials are valid, the SMS delivers the resource request to the Scheduling Service which interacts with the RMS to check the availability of the requested resource. If the requested resource is available, the Scheduler

invokes its allocation on the corresponding Resource Provision Service; otherwise, if the resource is not available, or by its nature can be provided concurrently by many Peers, the SCS starts searching for remote resources, transparently to the user. The Routing Service computes the destination(s) and sends request message(s). In the meantime, the Resource Provision Service provides the available local resource to the requester, through the CS.
The service migration framework described in chapter 2 has been implemented within SP2A, thus providing a solution for the discussed Mobile Service Problem.

3.1.8 Mobile Service Problem and Pull Solution

The Mobile Service Problem (illustrated in figure 3.10) is a *weak mobility* problem, because the execution unit that searches the service is dynamically bound to the code coming from a different site, but no migration of execution state is involved. We created a *Code on Demand* architecture, where a node accesses the service it needs, obtains the service know-how and executes it within its own context and resources.

Before downloading the service, the requesting peer must check to have enough resources to execute it, to avoid network overload. The service provider exposes an XML file with minimal hardware (number of processors, memory) and software (OS type and architecture, Java version, external libraries) requirements for service execution. This file is converted to a string and appended to the *description* field of Resource Provision Service (RPS).

The Pull solution implemented for Mobile Service problem involves the following steps:

- A peer node P_1 searches for a service σ
- The service is found on peer P_2
- P_1 decides to download the *.jar* and .wsdl files of service σ
- P_1 locally activates service S and starts to provide it

When the download has been completed, the requesting node shares the service in the peer network and becomes provider for it.

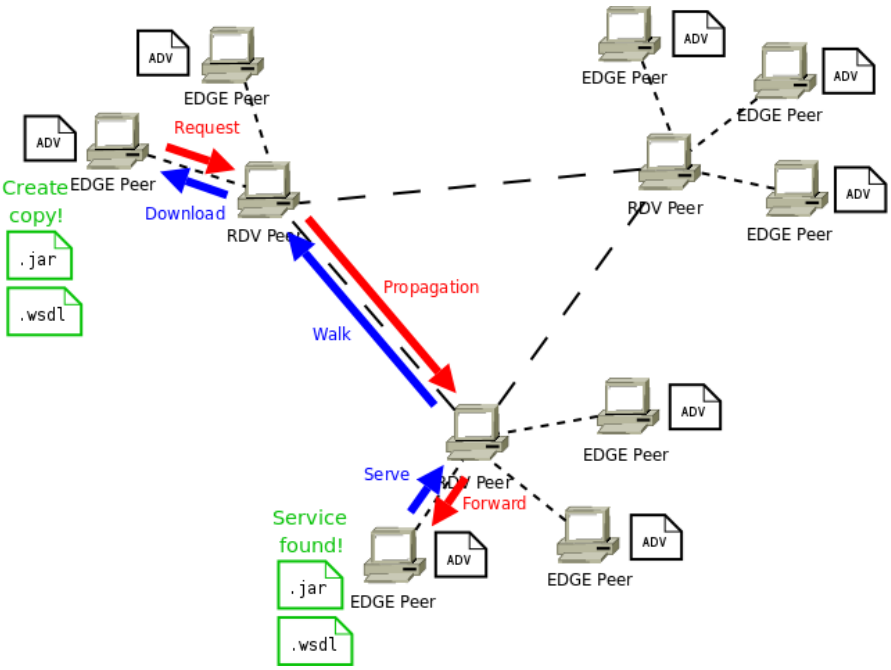

Figure 3.10: The Mobile Service Problem.

Communication protocol

SP2A supports JXTA technology for peer-to-peer message routing; JXTA protocols provide three transport mechanisms for communication among peers: *Endpoint Service*, which is a point to point communication level, *Pipe Service*, which integrates the Endpoint Service with the abstraction of virtual communication channel and *Jxta Socket*, at the higher level, that provides an interface similar to standard sockets' one. We used JXTA Sockets because they allow bidirectional communications and grant security to them; moreover they add to the message an ACK_NUMBER that provides additional information to the payload to assure reliability to the communication and correctness of sequentiality at the the receiver side. It is possible to configure the socket output buffer to reduce the number of packets and facilitate data reading operations.

The communication protocol for mobile Service Problem is described by the following algorithm (and illustrated in figure 3.11):

- the server, after deploying the service, waits for client connections through *JxtaServerSocket* class
- a client connects to the server which provide the discovered service through *JxtaSocket* class
- both client and server create an I/O stream for data exchange
- the client requests to download the service
- the server communicate the .jar file dimension and starts to send packets of bytes
- the client receives the packets and is able to reconstruct the file; then it sends a notification message to the server
- the server closes the connection and waits for other requests
- the client shares the service for download

Download Protocol

During the discovery phase, results are stored in a `DiscoveredRPSs` vector in `RPSManagerImpl` class; the dimension of such a vector gives information about the number of nodes found on a single discovery operation that are sharing the service for download: the greater the dimension is, the more the service is available in the network. The basic idea of our download protocol is to first try to download the service from the nodes listed in the actual vector and, if the vector is empty, or none of the peers are available at the moment, to invoke `findRPS()` method to start a new discovery in the JXTA peer-to-peer network.

Every time `downloadRPS()` method is invoked with the vector dimension as a parameter, a certain number of attempts to download the file from each provider is

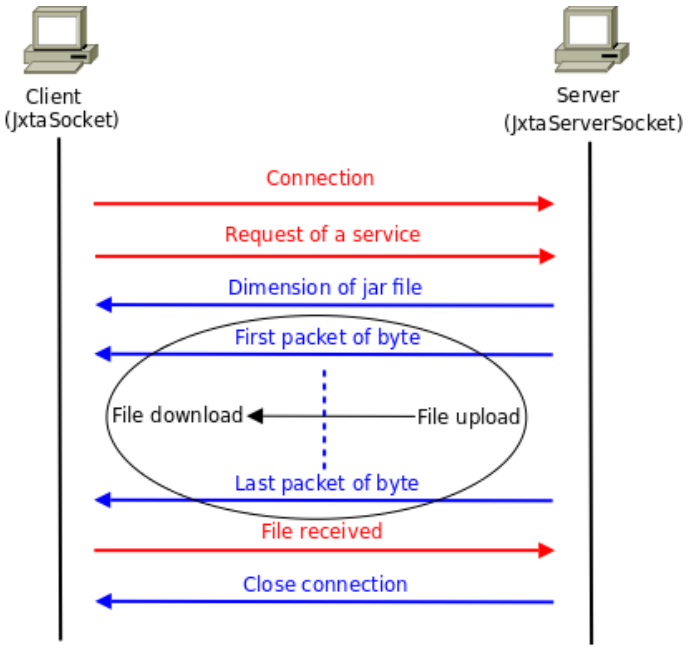

Figure 3.11: Service download protocol in SP2A.

performed, with an increasing timeout. The number of attempts increases if the discovery results in the vector are scarce, and decreases if a lot of providers are found in order to have a greater probability to find an available node from which downloading the service.

In the implementation of Pull solution for Mobile Service Problem, Java Reflection API is used to add in the runtime classpath of client the base class name of the downloaded service, and then to instantiate an object of the same class.

SP2A architecture currently enables users to actively contribute to the provision of resources in the peer-to-peer network, following a PaaS model, in which services are provided and diffused according to the users' requests. We have considered different applications that may take advantage of such an architecture, spacing from *P2P Video Streaming* to *e-learning communities* to Service Oriented Mobile Peers for geo-localization applications (developed in the context of a collaboration with

NATO 3C agency).

3.1.9 P2P Video Streaming

The traditional client/server model for video streaming is not very scalable with the increasing number of users, and introduces problems due to limited bandwidth and excessive workload at the server side. Peer-to-peer paradigm offers a good solution to the problem of load balancing; each peer, in fact, share its resources (bandwidth for communication among nodes and disk space for storing video stream data) with other peers that are requesting a service. Video streaming can be classified in two main categories: *live* and *on-demand*. In a live session the video is provided at real-time to all requesting users, and reproduction is synchronized; a *Video-on-Demand (VOD)* service allows to reproduce a video from any point and at any time, on the basis of users' requests. We considered the case of live video streaming as an example for the proposed SP2A-based architecture, because of the higher probability that peaks of requests occur (*i.e.* in correspondence of sport matches or concerts). In this context it is important to evaluate the dynamics of the system and the capability of adapting to the incoming requests for a service. A possible scenario for the P2P Video Streaming example is the following (as illustrated in figure 3.12):

- In an initial situation, M peers provide the video streaming service and K peers may host a video stream buffer, thus becoming provider too.
- A node starts a discovery for a certain video and selects one of the M provider peers; the selected node may accept the request only if it has not reached the limit of L served peers, otherwise requestor has to choose another provider from the list.
- The M provider nodes communicate with each other, and are able to estimate the global workload on the system, all the time. When the system appears not to be able to assure the required QoS, they request to the K available nodes to download the service and start providing it (such kind of interaction can be interpreted as ”solicited pull”).

- If a new provider peer is idle for a certain period of time, it may decide to remove the service and stop providing it. Otherwise, if requests for a service continue to increase it may also offer the service code for download, thus contributing to the service diffusion in the p2p network.

3.1.10 Peer-to-peer e-learning communities

In general, P2P is useful where shared resources and services lie at many endpoints. In the context of education this trend may enable each user to be much more proactive in the creation of paths for his/her own training and in communicating with other members which share some interests to build new group knowledge. Given that nowadays technology is mobile, students turn "nomad" [49], *i.e.* they overcome the boundaries of a classroom or a course to organize themselves in mobile groups which can quickly change their members and their goals. Especially in higher education, an institution should provide an IT infrastructure to involve learners in making meaningful connections to resources or other people.

A P2P e-learning community (PEC) is an unmoderated environment in which informal knowledge exchange, rather than formal training, takes place. As peers join the network, opportunities for more information to be stored, accessed, exchanged, and learned increase. A community of practice (CoP) is made by people who have a common interest in some subject and collaborate over an extended period to share ideas, find solutions, and build innovations [50]. The CoP concept refers as well to the stable group that is formed from such regular interactions. On the contrary, a PEC is a highly dynamic collection of peergroups, whose members collaborate over short periods to exchange knowledge. Mobility introduces some additional uncertainties on the stability of learning groups: mobile devices can have some limitations of computational and storage resources, or a low quality of achievable data connections, so mobile peers can unexpectedly become unreachable. To provide learning tasks without service interruptions the system infrastructure should be able to discover other peers able to support the learning activity. Moreover, the learning framework should be able to adapt service provision on the base of user context, both in term of his/her

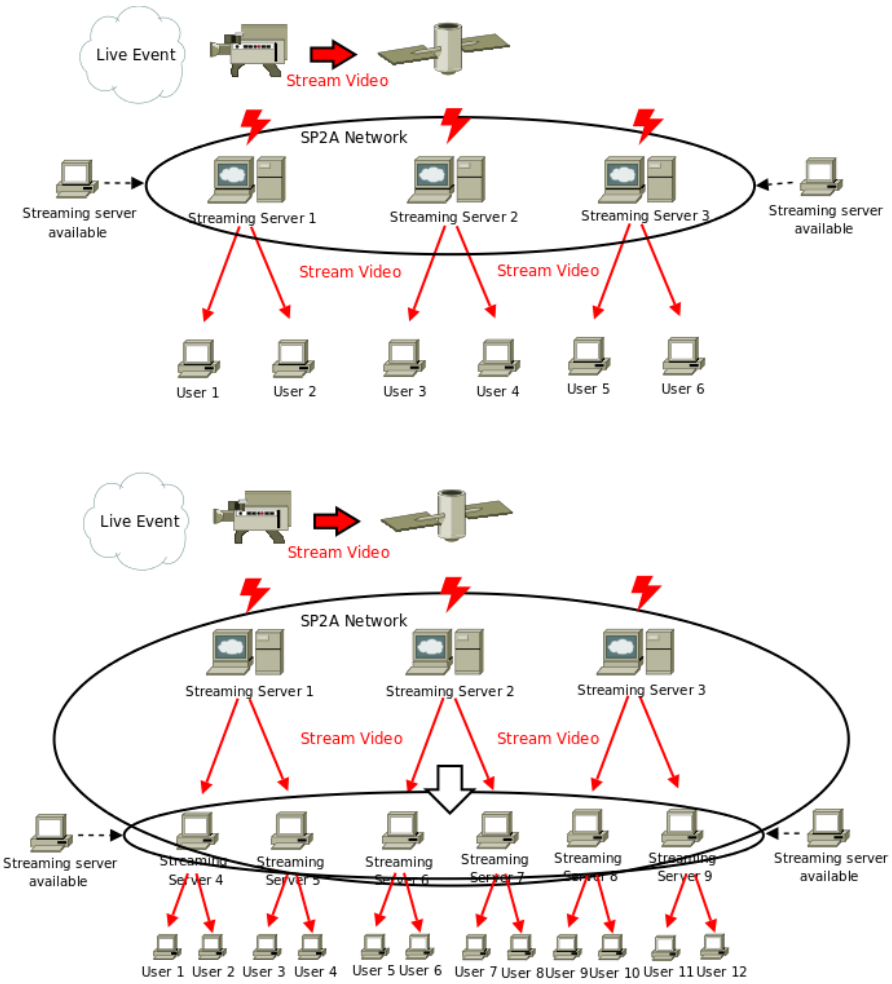

Figure 3.12: P2P Video Streaming Service in SP2A. In the first picture the service is offered by a limited number of servers, because the number of requests for it is low; as the number of requests increase, SP2A architecture enables other available peers to become providers of the service.

physical space and time and in terms of user's device and network capabilities. SP2A framework supports PECs in which users adopt different devices and perform differ-

ent kinds of learning activities.

Context-aware e-learning services

A context-dependent service for mobile consumers is a service whose behavior and output can be adapted to user preferences, user location, and user resources (device capabilities, negotiated bandwidth). Different kind of e-learning services can be provided in a context-aware fashion, using the P2P paradigm. Previously we referred to P2P as the synonym of e-learning services offered both by institutional actors, i.e. teachers, and by informal actors, *i.e.* students. At the infrastructural level, P2P means a set of mechanisms supporting user connectivity and, exactly, context-awareness. User-to-user services like video conference are highly demanding in terms of bandwidth and resilience. To cope with their requirements, P2P streaming techniques (bandwidth sharing, etc.) are a meaningful solution. With respect to centralized solutions, P2P streaming models are characterized by the lack of single points of failure, which is an advantage, but also by the topological mismatch between the overlay network and the physical network, which leads to non-optimal resource exploitation. On-demand services, which are not characterized by user-to-user live interactions, may be based on P2P discovery mechanisms and content-adaptation and replication techniques. We consider here two examples: on-demand streaming, and on-demand learning path construction. On-demand streaming applications with distributed providers of multimedia objects can be realized by means of traditional P2P architectural models for file sharing. But, with respect to this kind of application, on-demand streaming requires more guarantees on data flow continuity and delays. For this reason, data chunks scheduling must consider a broader range of parameters, which are often related to the characteristics of the underlying physical network. Moreover an efficient recovery mechanisms must be provided in order to guarantee the continuity of the data flow. The on-demand learning path construction service accepts queries and searches for related learning objects (LOs), building a learning path which can be dynamically rearranged during the process if the owners of some LOs leave the network. By definition LOs encapsulate both learning content and appropriate descriptive information (metadata). As illustrated by [51], semantic relationships

of LOs have essentially two spaces: the inner space, which implies the LO structure, and the outer space, which delivers the learning value of LOs within specific context of use. Both kinds of relationships enable the specification of learning paths, as sequences of semantically interrelated LOs. The e-learning architecture proposed by [51] consists of a portal, in which instructors publish LO descriptions and define semantic relationships between published LOs. Learners can browse or query the LOs of the portal, but also insert new LOs or enrich the descriptions of existing LOs in order, for example, to extend the available learning paths for a course. Our approach considers an even more dynamic environment, in which LOs are frequently updated (e.g. a student in mathematics may publish its exercise solutions day by day) and their availability depends on their replication degree (RD) in the overlay network (since there is no central repository). For example, we assume that a learner is interested in "Computer Science" for a half-day learning session. The learning path constructor suggests the following LOs:

Current time: 9AM

Proposed learning path:

1. **"Information theory (basics)" - OD - lecture slides - [RD=2] - time: n/a**
2. **"From finite state automata to Turing machine" - OD - lecture video - [RD=5] - time: 1h**
3. **"Algorithms and Data Structures" - U2U - live lesson - [RD=n/a] - time: 11AM-12AM**

It is important to observe that user-to-user services can figure as steps of a learning path. Their position in the LO sequence depends on the declared availability of the service provider.

Using SP2A middleware, we developed an application that can be executed both on PCs and PDAs or smart phones. It is very useful for learners to have a graphical interface that shows all the session steps and helps them to correctly manage the learning material.

The Learning Path Construction Service (OLPCService) allows the creation of a learning paths combining the Learning Objects deployed in the Sp2A network by

different Learning Object Provision Services (LOPService). These services expose methods that , given a set of prerequisites, return a list of LOs. The user/student has to provide three pieces of information to the system:

- the subject (e.g. Computer Science),
- the user's know-how about this subject (prerequisites),
- the time the user wants to spend studying.

In a typical scenario, the user/student with a mobile device (edge node) sends the request to the OLPCService (rendezvous node), which invokes the LOPServices (edge nodes) to obtain the LOs.

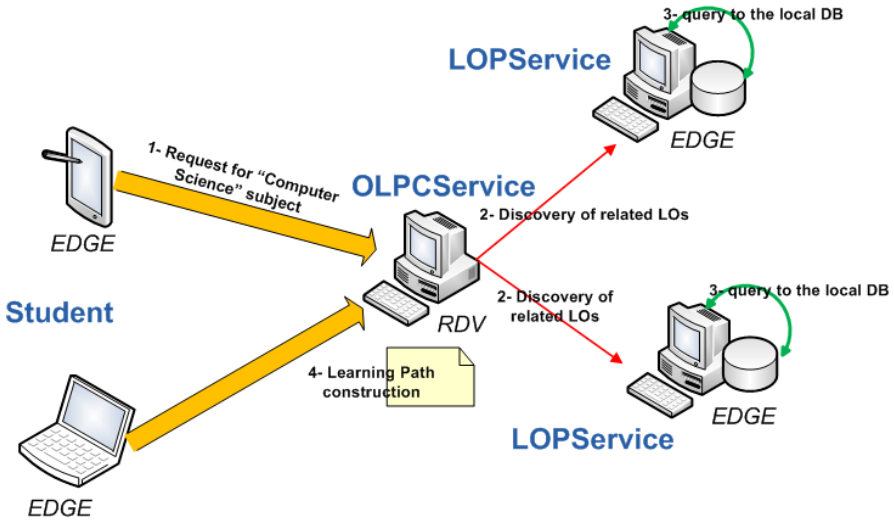

Figure 3.13: General structure for the OLPCService

The discovery of LOs is based on the name of the corresponding subject and the type of resource is selected from a dictionary, organized in order of importance (i.e. streaming video, slide, video on-demand, exercises, paper); during the discovery phase, preconditions are taken into account by selecting only LOs that are compatible with the user's knowledge. Each LOPService has a local Database with information

about all its available LOs. Once the OLPCService has received the complete list of learning objects, it proceeds ordering them in a timeline according with the described priority and avoiding overlapping of resources.

The client (*i.e.* student) runs a GUI-based application, that is composed of tab panels, with the capability of easily switching from one to the other even using a mobile device; in particular, in the case of mobile users, it's the application itself that recognizes the type of selected LO and runs the right program for it. When the execution is completed, it is possible to come back to the list of available LOs, and select another step of the learning path.

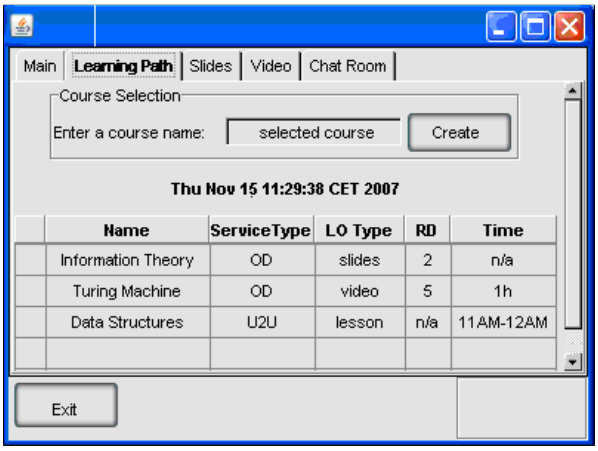

Figure 3.14: Learning path management panel

The user can join a peergroup, selecting from the list of available ones, and collaborate in exchanging knowledge and learn with other members. The Learning Path panel in figure 3.14 allows to select a topic and shows the corresponding learning material. For example, a learner who is interested in Computer Science" for a half-day learning session can type the course name in the learning path panel, and start the search. The learning path constructor suggests some LOs depending on the current time of request, and the GUI-based network explorer shows them in a table with all the information the learner needs to use them. The LO table shows if the service is on-demand or user-to-user, and what is the type of the LO, *e.g.* a presentation, a

live lesson video, etc. The replication degree is also provided, to inform the learner about the resource availability in the overlay network. The last information provided is about time: for on-demand services it could be the duration of a lesson or a video, while for user-to-user services it could show the availability of providers. For example if a teacher is available in the chat room at a particular time, the corresponding LO is scheduled depending to that time, that is shown in the table. Finally, the learner can select a LO and switch to the corresponding tab panel. From each panel it is possible to perform actions depending on the LO type: for example, the slides panel lists all the downloaded presentations, and it is possible to select one of them and show the slides. If a mandatory step in the learning path is jumped, a dialog will suggest the user to go back to the LOs list and select the right activity to successfully complete the learning path.

In the context of NAM, learning objects can be considered as atomic services provided by LOPServices and dinamically composable to create learning paths without involving OLPService, thus enabling a completely decentralized network. Moreover, code mobility mechanisms can be used for migrating resources when their requests increase for limited periods of time, *i.e.* the approaching of the exam date for a certain subject.

3.1.11 SOP application

In the context of a collaboration with NATO C3 agency in their WS-SP project, we have developed a prototype Service Oriented Peer (SOP) application for mobile devices with Java Micro Edition (J2ME) and Connected Device Configuration (CDC); in particular, devices must be compliant with CDC1.1/Foundation1.1/Personal Profile1.1.

The system for which SOP prototype has been designed allows geo-localization of users and exchange of other information related to position; four types of participants are involved (figure3.15:

- Mobile MIDP 2.0 compliant (GPRS). They provide tagged pictures with GEO position and receive and depict GEO information on the area using Web Ser-

vices.

- PDA CDC compliant (Wireless LAN). They receive and depict GEO information on the area using Web Services.
- KML Web Service server (LAN). It provides GEO KML ship ID and coordinates via Web Service in a given area.
- SI Search server (LAN). It provides GEO KML detailed ship information via Web Service and other type of GEO information related to a given area.

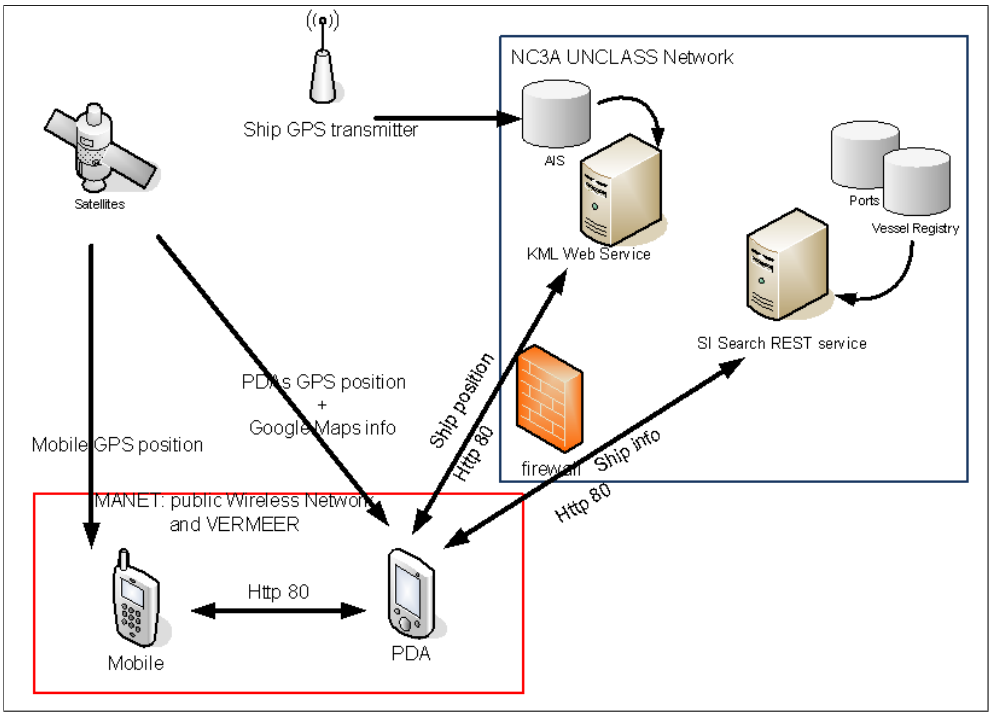

Figure 3.15: The system designed for the NC3A WS-SP project.

In the scenario NATO C3 agency has considered, a user with its PDA device accesses the network through Satellite communication GPRS. The PDA has the WS-SP software installed and is GPS receiver.He sees the his GPS location, selects two

points in the PDA screen to determine a screen square location. Then he uses the WS-SP application to find out Web Services in his MANET. The software will detect web services published by other members. He selects the web Service `infoShip` and press button select; in the next panel he enters the coordinates as parameters and selects the option get results in results panel. He goes to the results panel and verifies result is correct, and that it get the URL where the result is stored. He copies the URL and executes Google Maps for PDA that will depict the GEO points that represent the results of `infoShips` web service call. The user may also take a picture of Area Of Interest (AOI) and tag it with simple text; then he uses the WS-SP software to list WebServices which can receive field pictures and invoke one of them to send alive picture with the comment.

SOP application is based on SP2A and allows to manage three kinds of services:

- *Local Services* that are deployed locally in the peer
- *Remote Services* that are deployed remotely in other peers accessible on the SP2A/JXTA network
- *External Services* which are deployed in an application server (not part of the SP2A/JXTA network) either running on the same node hosting the peer or on an external host.

In particular, the application allows users to publish advertisements of local and external services in the SP2A/JXTA network, search the SP2A/JXTA network for available services, list external services deployed on an application server whose IP address is known and invoke discovered services, regardless of their external or remote type .

The SOP GUI shows different tabbed panels, and from the mobile device it is possible to switch from one to another using the cursor or the arrows where available.

- The first panel (Main) shows the joined peer groups; actually the only group available is Default, but it is possible to add others.
- The Local panel show services that are deployed locally in the peer, so it can be empty if no services are available. There are a table that lists all the services and

a Share Service button that allows to publish the advertisement of a selected service.

- The External panel lists services that are available on an external application server; to search for Services, type the service address (it must be a correct URL) in the text field and press the Show button; if any service is available at the selected address, the table will be filled with a list of services
- Remote panel is for services which are deployed remotely in the peer network; it is possible to search for services on other peers and to select one of them from the list (as for external services), in order to see all the operations the service offers, which are shown in the operation tab. To find a service in the peer network, it suffices to put the service name in the text field and a discovery is started in the SP2A network; the service table is then updated with all found services.
- Operation panel shows all the operation available for a selected service.
- Invocation panel is where the user introduces the required parameters for service invocation; it shows what service and operation have been selected and lists all the parameters. The user has to introduce a value for the required parameters and to select where to save the results, whether in a file stored locally or in the Result tab. Only when the result destination is selected the Invoke option is enabled.
- Result panel is filled with service response, in case the user has chosen to show the invocation result in the corresponding tab (figure 3.16)

In such a scenario, as well as in emergency management applications, network connectivity is not continuously available and only a few devices are enabled to communicate with the central station or to access the network through Satellite communication GPRS/UMTS. In order to preserve battery and CPU resources of devices that are in charge of communication, all the information obtained from the site (images, videos, GPS coordinates) can be distributed among other nodes ih the ad hoc network

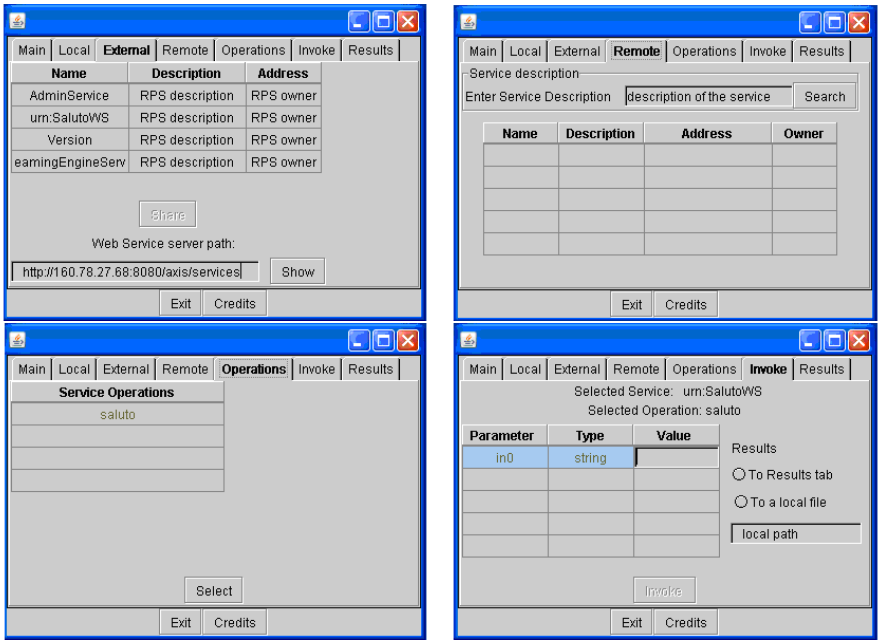

Figure 3.16: Panels for service discovery and invocation in SOP application

for elaboration, then composed as NAM resource and sent back to the communicating node.

3.2 Interoperability among heterogeneous WS platforms: STIL project

STIL ("Strumenti Telematici per l'Interoperabiliá delle reti di imprese: Logistica digitale integrata per l'Emilia-Romagna", *i.e.* telematic tools for interfirm networks interoperability: digital logistics for the Emilia-Romagna region) is a regional project in the field of IT applied to logistics I had been involved with in the first period of my Ph.D. activity. Logistics can be defined as the geographical repositioning of raw materials, work in process, and finished inventories where required at the lowest cost possible, through the integration of information, transportation, inventory, ware-

housing, material handling, and packaging. In STIL, logistics is considered in the context of interactions between providers and consumers of products and services in the e-market place. STIL's final goal is to create a region-wide *Virtual Logistic Pole* providing Enterprise Application Integration (EAI) for business to business (B2B) applications to manufacturing firms, transportation carriers and logistic hubs. The concept of value-chain is applied not only to the manufacturer-carrier chain, but also to public interest, with particular emphasis on process observation and optimization for environment and quality of life safeguard.

Key elements for STIL architecture development are:

- Domain business model: description of processes and of static and dynamic elements of the value chain
- Data model: a Global Virtual View (GVV) has been defined which represents a unified language for information exchanged in the logistic processes.
- Software architecture: a set of base software components to allow interoperability among etherogeneous systems (STIL Platform).
- Pilots: a set of application prototypes for logistics operations leveraging on STIL platform and STIL GVV.

Figure 3.17 illustrates the interaction between STIL actors, namely manufacturers, logistic operators and carriers, logistic centers. In a typical scenario, we can consider two types of services, as STIL application, which are critical for the value-chain:

- the `TransportationBroker` which accepts transport orders from manufacturers' logistic operators, requests and retrieves mission plans, interacts with carriers to obtain their quotations and finally orders the best one;
- the `MissionPlanner` which is invoked by the TransportationBroker to compute optimized mission plans considering different transportation systems. In this scenario, each kind of service may be offered by multiple competing providers with different costs and implementations. In the prototype, emphasizing decentralized and direct e-business interaction, NAM can provide to each node the basic layer for peer-to-peer service sharing and interaction.

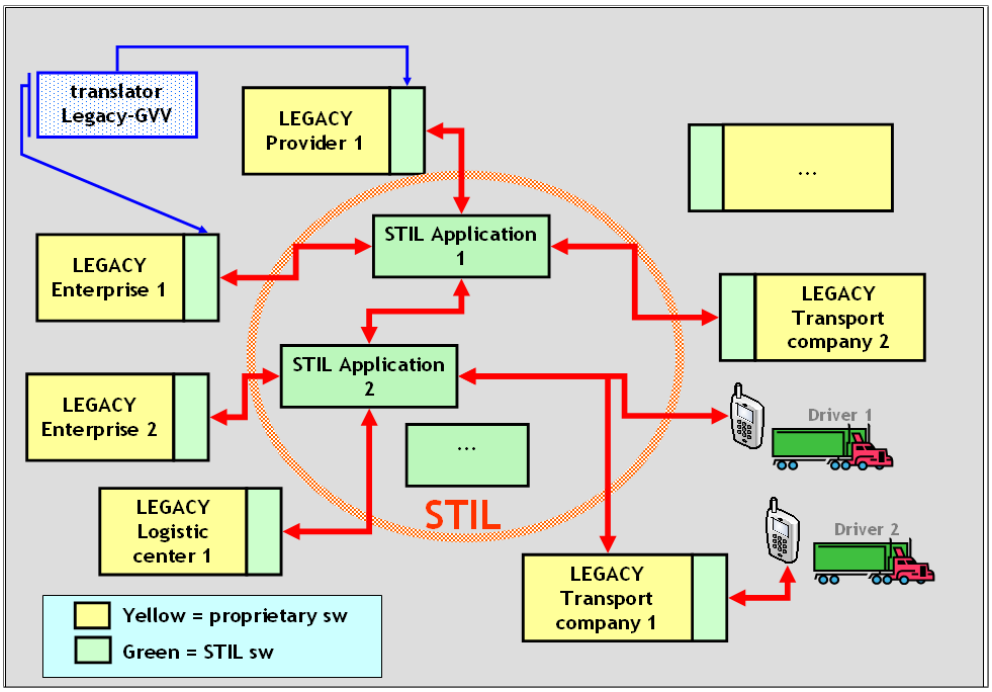

Figure 3.17: Interaction among STIL actors and applications

STIL defines an ICT infrastructure which offers mechanisms and policies for the semantic integration of applications (eServices) which realize stategical features for the value-chain. An eService is a software entity deployed by a service provider across the Internet. eServices can be statically selected by subscribing off-line contracts, or dynamically discovered using several approaches (for example, the NAM approach). Obviously, an eService can be selected not only for its functionalities, but also for the quality of service (QoS) it guarantees. From the technological point of view, eServices are implemented with Web Service technologies. By focusing solely on messages, the Web Service model is completely language, platform and object model-agnostic. A Web Service can be implemented using the full feature set of any programming language, object model, and platform. A Web Service can be consumed by applications implemented in any language for any platform.

Our contribution to the project, as described in the following, basically involved

the development of a system for authentication of users accessing to the wireless network of a STIL-federated company and a client application for the use of STIL services from a mobile device (PDA).

3.2.1 Secure access to the STIL network

As represented in figure 3.18, the system we have developed for secure access to the wireless network enables the mobility of users among different sites within STIL federated security model. User's arrival in a new site does not require a 'SysAdmin' because authentication process is federated and is based on trust relations both direct or mediated by STIL platform.

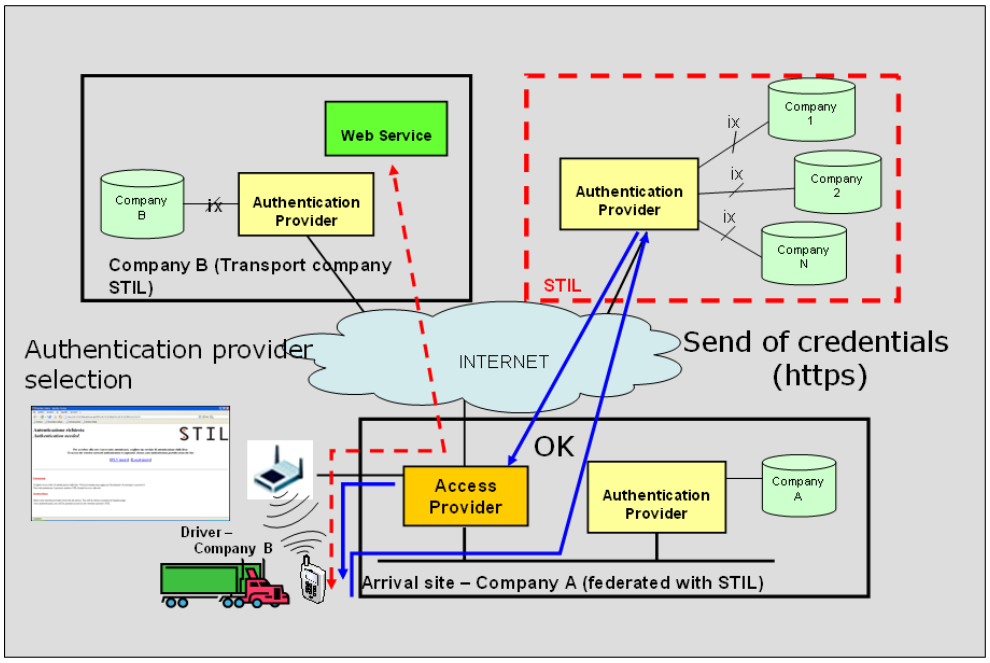

Figure 3.18: STIL authentication system for wireless netwotks

The proposed solution is designed for mobile devices and has a user friendly interface which is suitable for technically non-expert users. It is based on the use of

Captive Portal technology thus improving the end-to-end security level (from device to company system). A web interface is available for the user selection of authentication service among the following:

- accessing company (local)
- origin company
- STIL community (global)

3.2.2 Access to the LVP services from a mobile device

Using STIL infrastructural supports and J2ME APIs we have implemeted an application that allows a client (*i.e.* truck driver) to interact with the system of the transport company in order to receive a "Transport Mission" on his PDA and to send back periodic information from it (figure 3.19).

In details, the application enables the following operations:

- Obtain a new transport mission from the transport company system, or load a previously stored one;
- Visualize details of a transport mission;
- Send an event of enter/exit in relation to a site;
- Send an exception event (*e.g.* truck break down).

All interactions are allowed through Web Services invocation; with this application the transport company system receives information about the transport mission completion directly from the driver and in real time.

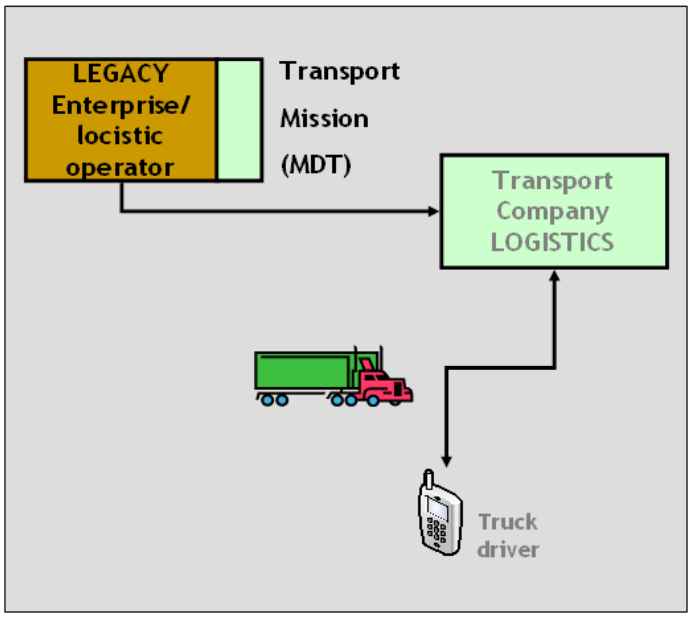

Figure 3.19: Interaction between driver and transport company system within STIL infrastructure

Conclusions

The major achievements of this work are related to the definition of Networked Autonomic Machine (NAM), a formal framework to design distributed computing systems with shared resources, and to the development of the mobile version of JXTA-SOAP and SP2A middlewares.

A Networked Autonomic Machine (NAM) is a hardware/software entity that is programmed to be completely altruistic, providing resources to other NAMs. In a peer-to-peer system of NAMs, each node can act both as resource consumer and resource provider, and contributes to the effective and efficient functioning of the whole system. NAMs can be of different types and complexities, depending on the device and on the characteristics of the offered resources. Several kinds of devices are considered: PCs and workstations, notebooks, PDAs, smart-phones, as well as sensors and actuators. Devices can be classified on the basis of their system characteristics (OS, processor type, memory, I/O type, battery, connectivity) or their functionalities (camera, communicating, processing, sensors, etc.). The software layer of each NAM includes a lightweight control module implementing a peer-to-peer overlay scheme.

The service-oriented version of NAM namely *Networked Service-oriented Autonomic Machine (NSAM)* has been presented. In NSAM network, each NSAM provides atomic services and cooperates with other NSAMs to build composite services. An atomic service is defined as the minimal executable function unit, that cannot be decomposed and whose execution can transform a given state to another state. Atomic services provided by different peers can be statically or dynamically aggregated (proactively or on-demand) to realize new complex tasks. Both atomic and

composite services are tuples, the latter with a specific rule that allows to combine atomic services; this rule is represented as a directed workflow graph.

The presented modeling formalism NAM is particularly suitable for building peer-to-peer systems characterized by services that migrate on-demand, thus allowing to cope with highly dynamic environmental conditions. Within the context of the formal framework, we have explained how resources provided by each NAM may be shared and also migrated within a group of NAMs.

The definition and design of NAM framework has its basis in two projects that have been developed and managed by the Distributed System Group of University of Parma: *SP2A (Service-oriented Peer-to-peer architecture)*, a lightweight middleware enabling service-oriented peers for efficient and robust distributed applications, and *JXTA-SOAP*, a component which extends the JXTA middleware, with the design goal of wrapping Web Services in JXTA services through the use of JXTA for Web Service discovery and SOAP message transport. JXTA-SOAP has been included in SP2A middleware as an external API, thus enabling Web Services technology for the development of Service Oriented Peers (SOPs). We have presented the main features of both SP2A and JXTA-SOAP, focusing on the additional features we have implemented, enabling service migration mechanisms in the first and service provision from resource constrained devices in the latter. Moreover, transport level security mechanisms have been introduced in the J2ME version of Jxta-Soap, by means of the creation of new type of pipe, `JxtaUnicastCrypto`, by which it is possible to cipher message contents, and the definition of a new security policy, suitable for Connected Device Configuration (CDC) and Personal Profile.

Finally, we presented several applications, which have been implemented within SP2A and JXTA-SOAP, with emphasis on the enhancements that it is possible to obtain with the introduction of NAM framework. These applications target different fields, from Ambient Intelligence to e-learning communities, to emergency management scenarios, and finally to logistics, with the goal of enabling interoperability among heterogeneous platforms (including resource constrained devices) and ubiquitous sharing of services and resources.

Future work in NAM is related to the simulation of the complete system with

Discrete Event Universal Simulator (DEUS). DEUS is a general purpose simulator, developed by the Distributed System Group of University of Parma, which aims at becoming one of the reference tools in the field of complex system simulation. Complex systems are dynamic and composed of interconnected parts that as a whole exhibit one or more properties that could not be gathered from the properties of the individual parts. A network of NAM nodes can be considered a complex system, and can be represented by a simulation model, *i.e.* a specification of the system in terms of a set of *states* and *events*. Performing a simulation means mimicking the occurrence of events over time, and recognizing their effects as represented by states.

For asynchronous complex systems, characterized by events that are not guaranteed to occur at regular intervals, and by the lack of a bound on the time step (*i.e.* it should not be so small as to make the simulation run too long, nor so large as to make the number of events unmanageable), it is more appropriate to adopt an event-driven simulation [52]. Examples of such systems are distributed computing systems based on the peer-to-peer paradigm, with nodes randomly joining and leaving, but also emergency rescue and crisis management scenarios, where rescuers do not arrive and leave at regular time intervals. Since NAM framework belongs to this category of complex systems, simulation appears to be a useful approach to evaluate the system performances before fully implementing it.

After the completion of the simulation phase, a new version of SP2A, directly mapping on NSAM, will be implemented and tested on PlanetLab, a global distributed system that supports the development of new network services. Since the beginning of 2003, more than one thousand researchers at top academic institutions and industrial research labs have used PlanetLab to develop new technologies for distributed storage, network mapping, peer-to-peer systems, distributed hash tables, and query processing. The PlanetLab Consortium currently consists of 1056 nodes at 490 sites, and is managed by Princeton University, the University of California at Berkeley, and the University of Washington. Our University participates in the Consortium.

Bibliography

[1] M. Weiser. The Computer of the 21st Century. *Scientific American*, September 1991.

[2] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8:10–17, August 2001.

[3] J. Gaber. Spontaneous Emergence Model for Pervasive Environments. *IEEE Globecom Workshop*, November 2007.

[4] Reza B'Far. *Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML.* Cambridge University Press, 2005.

[5] Lionel M. Ni Pei Zheng. *Smart Phone Next Generation Mobile Computing*. Morgan Kaufmann Publishers, 500 Sansome Street, Suite 400, San Francisco, CA, 2006.

[6] F. Koushanfar, V. Prabhu, M. Potkonjak, and J. M. Rabaey. Processors for mobile applications. In *in Proc. 2000 IEEE Int. Conf. Computer Design: VLSI in Computers and Processors* , pages 603–608, 2008.

[7] R. Suoranta. New Directions in Mobile Device Architectures. In *in Proc. of the 9th Euromicro Conference on Digital System Design (DSD'06)*, 2006.

[8] T. Bodhuin, G. Canfora, R. Preziosi, and M. Tortorella. Open Challenges in Ubiquitous and Net-Centric Computing Middleware. *IEEE International Workshop on Software Technology and Engineering Practice*, September 2005.

[9] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Poyoul, and B. Yeager. Project JXTA 2.0 Super-Peer Virtual Network. Technical report, Sun Microsystems, 2003.

[10] OSGi Alliance. Osgi: the dynamic module system for java [online]. Available from World Wide Web: `http://www.osgi.org`.

[11] OASIS. Web Services Resource Framework (WSRF) v1.2. Technical report, April 2006.

[12] Distributed Systems Group and Sun MicroSystems. Jxta-soap project [online]. Available from World Wide Web: `https://soap.dev.java.net`.

[13] A. Harrison and I. Taylor. WSPeer - An Interface to Web Service Hosting and Invocation. In *Proc. of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, May 2005.

[14] S. N. Srirama, M. Jarke, and W. Prinz. A Mediation Framework for Mobile Web Service Provisioning. In *Proc. of the 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW'06)*, October 2006.

[15] S. N. Srirama, M. Jarke, and W. Prinz. MWSMF: a Mediation Framework Realizing Scalable Mobile Web Service. In *Proceedings of Mobilware'08*, February 2008.

[16] S. Berger, S. McFaddin, C. Narayaswami, and M. Raghunath. Web Services on Mobile Devices - Implementation and Experience. In *Proceedings of the Fifth IEEE Workshop on Mobile Computing Systems & Applications*, October 2003.

[17] R. A. van Engelen and K. Gallivan. The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. In *Proc. of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid2002)*, pages 128–135, May 2002.

[18] Microsoft. .net compact framework [online]. Available from World Wide Web: `http://msdn.microsoft.com/en-us/netframework/aa497273.aspx`.

[19] Sun MicroSystems. J2me web services apis (wsa) [online]. Available from World Wide Web: `http://java.sun.com/products/wsa/`.

[20] Sun MicroSystems. Jsr 172: J2me web services specification [online]. Available from World Wide Web: `http://jcp.org/en/jsr/detail?id=172`.

[21] S. Haustein and J. Seigel. ksoap2 project [online]. Available from World Wide Web: `http://ksoap2.sourceforge.net`.

[22] P. Plebani. mas project [online]. Available from World Wide Web: `https://sourceforge.net/projects/masproject`.

[23] Helal S., Mann W., El-Zabadani H., King J., Kaddoura Y., and Jansen E. The Gator Tech smart house: A programmable pervasive space. *IEEE Computer*, pages 64–74, 2005.

[24] OSGi Alliance. Amigo project. ambient intelligence for the networked home environment [online]. Available from World Wide Web: `http://www.amigo-project.org`.

[25] Gu T.AND Pung H.K. and Zhang D.Q. A middleware for building context-aware mobile services. In *in Proc. of IEEE Vehicular Technology Conference*, pages 2656–2660, 2004.

[26] Persona project web site [online]. Available from World Wide Web: `http://www.aal-persona.org`.

[27] Baronti P. AND Pillai P. AND Chook AND V.W.C.AND Chessa S.AND Gotta A. AND Hu Y.F. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Computer Communications*, 30(7):1655–1695, 2007.

[28] Fides-Valero A., Freddi M., Furfari F., and Tazari M.-R. The PERSONA framework for supporting context-awareness in open distributed systems. In *in Proc. of Ambient Intelligence conference (AmI08)*, pages 91–108, 2008.

[29] M. Kumar, B.A. Shirazi, S.K. Das, B.Y. Sung, D. Levine, and M. Singhal. Pico: a middleware framework for pervasive computing. *Pervasive Computing, IEEE*, 2(3):72–79, July-Sept. 2003.

[30] S. Kalasapur, M. Kumar, and B.A. Shirazi. Dynamic service composition in pervasive computing. *Parallel and Distributed Systems, IEEE Transactions on*, 18(7):907–918, July 2007.

[31] Xiaohui Gu, K. Nahrstedt, and Bin Yu. Spidernet: an integrated peer-to-peer service composition framework. In *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on*, pages 110–119, June 2004.

[32] Paolo Bellavista, Antonio Corradi, Rebecca Montanari, and Cesare Stefanelli. Context-aware middleware for resource management in the wireless internet. *IEEE Transactions on Software Engineering*, 29:1086–1099, 2003.

[33] S. Kalasapur, M. Kumar, and B.A. Shirazi. Dynamic service composition in pervasive computing. *Parallel and Distributed Systems, IEEE Transactions on*, 18(7):907–918, July 2007.

[34] A. Carzaniga, G. P. Picco, and G. Vigna. Designing Distributed Applications with Mobile Code Paradigms. In *ICSE - International Conference on Software Engineering*, April 1997.

[35] A. Carzaniga, G. P. Picco, and G. Vigna. Is Code Still Moving Around? Looking Back at a Decade of Code Mobility. In *ICSE - International Conference on Software Engineering*, May 2007.

[36] S. Haustein and J. Seigel. ksoap2 project [online]. Available from World Wide Web: `http://ksoap2.sourceforge.net`.

[37] Krishna A., Schmidt D. C., and Stal M. Context Object: A Design Pattern for Efficient Middleware Request Processing. In *in Proc. of the 12th Pattern Language of Programming Conference*, 2005.

[38] Gamma E., Helm R., Johnson R., and Vlissides J. *Design Patterns*. Addison-Wesley, 1995.

[39] I. Pyarali, M. Spivak, R. Cytron, and D. C. Schmidt. Evaluating and Optimizing Thread Pool Strategies for Real-Time CORBA. In *in Proc. of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001)*, 2001.

[40] Govoni D. and Soto J.C.. *JXTA and security. In JXTA: Java P2P Programming*. Sams Publishing, 2002.

[41] Mikey: Multimedia internet keying [online]. Available from World Wide Web: `http://www.ietf.org/rfc/rfc3830.txt`.

[42] Mikey-rsa-r: An additional mode of key distribution in multimedia internet keying (mikey) [online]. Available from World Wide Web: `http://www.ietf.org/rfc/rfc4738.txt`.

[43] Haddow G.D. and Bullock J.A. *Introduction to Emergency Management*. Butterworth-Heinemann, 2004.

[44] Commission of the European Communities. Communication on Reinforcing the Union's Disaster Response Capacity. Technical report, 2008.

[45] Commission of the European Communities. Communication on Global Monitoring for Environment and Security (GMES): Establishing a GMES capacity by 2008. Technical report, 2004.

[46] Gallup Organization. General public survey on the European Galileo Programme. Technical report, 2007.

[47] M. Amoretti, F. Zanichelli, and G. Conte. SP2A: a Service-orientd Framework for P2P-based Grids. In *Proceedings of the 3rd international workshop on Middleware for grid computing*, pages 1–6. ACM Press, 2006.

[48] M. Amoretti, M. Reggiani, F. Zanichelli, and G. Conte. Peer: an Architectural Pattern. In *Proceedings of the 12th Pattern Languages of Programs (PLoP)*, pages 1–14, 2005.

[49] A. Bryan. Going Nomadic:Mobile Learning in Higher Education. *EDUCAUSE Review*, 39(5):28–35, 2004.

[50] E. Wenger. *Communities of practice - Learning, meaning and identity*. Cambridge University press, 1998.

[51] D. Kotzinos, S. Pediaditaki, A. Apostolidis, N. Athanasis, and V. Christophides. Online Curriculum on the Semantic Web: The CSD-UoC Portal for Peer-to-Peer E-learning. In *in Proc. of the 14th international conference on World Wide Web*, pages 307–314, 2005.

[52] B. Zeigler, T. Kim, and H. Praehofer. *Modeling and Simulation*. Academic Press, 2000.

Acknowledgments