A. D. MCCXXIV

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Dottorato di Ricerca in Ingegneria Informatica ed Automatica

Comunità Europea
Fondo Sociale Europeo

SERVICE DISCOVERY AND DELIVERY IN INTEROPERABLE NOMADIC COMPUTING SYSTEMS

CRISTIANO DI FLORA

Tesi di Dottorato di Ricerca

Novembre 2004

Dipartimento di Informatica e Sistemistica

via Claudio, 21- I-80125 Napoli - ☎ *[#39]* (0)81 768 3813 - *[#39]* (0)81 768 3816

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Dottorato di Ricerca in Ingegneria Informatica ed Automatica

Comunità Europea
Fondo Sociale Europeo

SERVICE DISCOVERY AND DELIVERY IN INTEROPERABLE NOMADIC COMPUTING SYSTEMS

CRISTIANO DI FLORA

Tesi di Dottorato di Ricerca

(XVII Ciclo)

Novembre 2004

Il Tutore
Prof. Stefano Russo

Il Coordinatore del Dottorato
Prof. Luigi P. Cordella

Il Co-Tutore
Prof. Kimmo Raatikainen
(University of Helsinki)

Dipartimento di Informatica e Sistemistica

via Claudio, 21- I-80125 Napoli - ☎ *[#39]* (0)81 768 3813 - *[#39]* (0)81 768 3816

"You are never given a wish without being given the power to make it true.
You may have to work for it, however."
(Richard Bach, *"Illusions, The Adventures of a Reluctant Messiah"*)

Table of Contents

List of Tables

List of Figures

Acknowledgements

The first lines of this thesis should be words of gratitude to all those who helped to make this thesis possible. First of all, I have to thank professor Stefano Russo and his faithful assistant Domenico Cotroneo for being patient reviewers and helpful guides of my research works from the very early life of my research career (i.e., my Master Thesis) and for supporting this work with ideas, criticism, and crucial insights. Some valuable ideas for this research were inspired by a tight interaction with professor Kimmo Raatikainen and his research group during my six-month stay in Finland as a visiting student and researcher. His hints and insights gave a significant contribution to this thesis. His kindness merged to the friendliness of Tiina Niklander and of Oriana, Simone, and Davide made my first finnish winter more friendly and warm: I will never forget my first "Joulukahveja" at the Computer Science department with them, and I hope we'll keep on doing so in the next years.

Of course, no one could survive the perils of a PhD without the friendship of the several companions who share in the misery and joys and forced me to have fun in spite of my research-addiction as well. Thanks to all the PhD students and researches with whom I had the pleasure to collaborate (and to joke) during the last years.

Much appreciation goes to the rest of my family; their support and belief in me have always inspired me in my endeavors. Last, but certainly not least, I thank my sweet-heart whose love gave me the strength and encouragement to follow the path that I felt was right, and whose patience and unselfishness allowed me the freedom to complete the task. This dissertation is lovingly dedicated to her and to all the people who love to chase dreams fearlessly.

Naples, Italy — Cristiano di Flora
November 30, 2004

Introduction

In the nomadic computing environment people interact with various companion, embedded, or invisible computers not only in their close vicinity but potentially anywhere; they currently see the resulting environment as the composition of heterogeneous sub-systems which support personal and nomadic applications. These sub-systems are called nomadic computing domains. They consist of communicating clusters of personal digital devices, devices shared with other people and even infrastructure-based systems. These domains may have an unrestricted geographical span, and incorporate devices into the nomadic environment regardless of their geographic location. In order to do this they need the services of infrastructure-based networks and ad-hoc networks to extend their reach.

Since the requirements of each nomadic computing domain are different from those of the others, the diversity of current technologies is the key factor to suit all the domains. Unfortunately, though the current service platforms succeed in supporting a particular nomadic computing domain, they are scarcely interoperable with each other, thus limiting the possibility to use them as building blocks for larger nomadic systems. Moreover, it is very unlikely that either *i*) one technology will fit all the requirements of a nomadic environment or *ii*) device manufacturers will embrace multiple service discovery and delivery technologies on mobile devices. This is especially

true with respect to low-cost and embedded devices, i.e., with respect to personal devices used by nomadic users. Hence, the challenge for future service provisioning systems will be the definition of an architecture for the effective interworking of existing service infrastructures, and that fulfills the needs of personalized, context-aware, nomadic services and solutions at the same time.

The aim of this thesis is to bring a significant contribution in this area, with the goal of enabling the definition of novel strategies to support interoperability between future nomadic users.

The contribution of this work is to provide a comprehensive and thorough description of how to effectively deal with the heterogeneity of both discovery and delivery infrastructures, and of specific applications as well.

As for the infrastructure, this thesis presents a novel architecture to address interoperability and scalability of both the discovery and the delivery of nomadic applications. It aims to mitigate the diversity of service representations, technologies, and interaction models of current nomadic computing domains. The proposed architecture allows to build a nomadic environment by composing nomadic computing domains (built upon current technologies), thereby allowing clients to discover and use services across domain borders. This architecture is composed of discovery and delivery agents: the former are in charge of connecting domains into a single logical domain, whilst the latter manage service sessions that users can establish to services after service discovery. The proposed solution uses a novel algorithm to decide which services to publish across diverse domains so as to reduce the burden due to service discovery across a huge number of diverse discovery platforms. This algorithm specifically uses both functional and technology-related constraints to drive the import

and export operations. The proposed solution is evaluated both by simulation and by the implementation of a prototype used to connect Jini-based to Bluetooth-based domains.

As for specific applications, two case studies are thoroughly described so as to show how to deal with unpredictability and context-awareness. This work specifically shows how to handle the nomadicity concerns of *i*) location-awareness for mobile devices and of *ii*) unpredictable system behavior for mobile multimedia services.

Chapter 1

Service Provisioning in Nomadic Computing systems

This chapter sheds some light on Nomadic environments. Specifically it firstly defines nomadicity and compare nomadic computing with several computing paradigms for mobile users. Subsequently two example scenarios for nomadic computing are described, which are used in the rest of the chapter to define the requirements of nomadic environments, as well as to identify issues in satisfying such requirements.

1.1 Nomadicity and modern mobile computing

Nomadicity is the tendency of a person, or a group of people, to move frequently. The widespread diffusion of personal devices is paving the way for services to be provided to nomadic users according to the anytime, anywhere access rule of Nomadic Computing, defined by Leonard Kleinrock in 1995 [31]. In other words, nomadicity is the ability to move easily from place to place and retain access to a rich set of information and communication services[1] while moving, at intermediate stops, and

[1]By services, I refer to a device (such as a printer, projector, or a scanner) or an application that provides useful functions to end-users.

at the destination. Its basic characteristics include location independence, device independence, motion independence, widespread access, and ease of use. As stated by L. Kleinrock, "the essence of a nomadic environment is to automatically adjust all aspects of the user's computing, communications, and storage functionality in a transparent and integrated fashion". This transparency [2] refers to the following variables: *i*) location user is at; *ii*) communication device and computing platform he is using; *iii*) current state of system resources (e.g., bandwidth, device features, available services); and *iv*) whether or not the user is in Motion.

As for nomadicity, different computing paradigms are required by mobile users. Such paradigms can be broken-down into the following categories:

Distributed computing: system is physically distributed. User can access system/network from various points which are tied to a physical location. E.g. Unix systems, world wide web, email; in this case the network is fixed, while the user moves;

Mobile computing: continuous access, automatic reconnection; in this case network topology changes, because each user moves across different locations; thus the main challenge is to deal with changing node location.

Pervasive computing: computing environment including sensors, cameras and integrated active elements; the goal is to make the computer invisible;

Wearable computing: computing devices integrated into bodily experience (smart clothing);

[2]The notion of transparency here refers to the perception of a computing environment that automatically adjusts to the processing, communications and access available at the moment.

Ubiquitous computing: computers are embedded in user's natural movements and interactions with pervasive computing environments; users move across several pervasive environments;

Nomadic (portable) computing: provide users with access to popular desktop applications, applications specially suited for mobile users, and basic communication services in a mobile, sometimes wireless, environment; in this case, the network is fixed, except for user devices and their point of attachment. Users can take their device with them, and decide to use different devices for different situations.

These computing paradigms have been topics of research since the early 1990s [23, 31, 36, 44, 63, 40]. Those topics are related by a desire to address mobility from different viewpoints. For instance, ubiquitous computing is quite similar to nomadic computing from a user's perspective, as they both concern with user and terminal mobility. However, this common desire makes it difficult to precisely classify existing systems. Confusion is further exacerbated by the usage of nomadic and personal devices either for ubiquitous or nomadic computing applications. However, the different strategy to deal with dynamic changes allows for distinguishing nomadic and ubiquitous systems: nomadic computing systems are mostly reactive, since they react to events or situations rather than acting first to change or prevent something [37]. Conversely, ubiquitous computing systems are proactive as they can take action (so as to prevent something) by causing change and not only react to change when it happens [52]. Another significant difference lies in the main requirement to satisfy: while ubiquitous computing systems mostly concern with the seamless integration

of novel user interfaces and computing devices into human life, nomadic computing systems deal with assemblage of interconnected technological and organizational elements, which enable physical and social mobility.

Since this work concerns with nomadic computing, in the following section several detailed examples of nomadic environments are described, so as to provide the reader with a more comprehensive analysis of their characteristics.

1.2 Nomadic computing scenarios

1.2.1 Introduction: The Nomadic Computing Environment

The future nomadic computing should cover all aspects of personal life. When interacting with computers today, we move in different computing domains. Communication and computing devices move; users move and change their devices; (sub)networks in cars, trains and airplanes move; software moves from one execution environment to another. In this subsection a future is envisaged where personalization and ubiquitous access to information and communication are merged into the so-called *Nomadic Computing Environment*.

The nomadic computing environment is composed of several heterogeneous discovery and delivery sub-systems. These systems are called nomadic computing domains. They constitute a category of distributed systems with very specific characteristics. They are configured in an ad hoc fashion, as the opportunity and the demand arise, to support personal and nomadic applications. Nomadic computing domains consist of communicating clusters of personal digital devices, devices shared with other people and even infrastructure-based systems. At the heart of a nomadic computing domain is a core Personal Area Network (PAN), which is physically associated

with the owner of the domain. Unlike the present PANs that have a geographically limited coverage, the Personal Operating Space, nomadic computing domains may have an unrestricted geographical span, and incorporate devices into the nomadic environment regardless of their geographic location. In order to do this they need the services of infrastructure-based networks and ad-hoc networks to extend their reach.

The concept of service is becoming crucial in Nomadic Computing, since end-users see such domain as a collection of interacting services, each providing a well-defined functionality.

Heterogeneity and diversity of NCDs have been addressed by adopting a Service-oriented approach so far. This approach suggests to build each NCD according to the *service oriented architecture*(SOA) model. A SOA allows clients to discover and use available services, thereby it is usually composed of service discovery and service delivery infrastructures. More specifically, it is composed of the following main entities: *i*) **service**, i.e., the logical entity, defined by one or more published interfaces; *ii*) **service provider**, i.e, the entity that implements a service specification; *iii*) **service requestor**, i.e., the software entity that requests a service to a specific provider (it can be an end-user application or another service); *iv*) **service locator**, i.e., a service provider that acts as registry and allows the lookup of service provider interfaces and service locations; and *v*) **service broker**, i.e., a service provider that forwards service request to one or more additional service providers. This model allows to design and deploy federations of services in which the kind and the number of services can dynamically vary[3]; the resulting services can be deployed according to the following paradigms: *i*) service implemented on a single machine, *ii*) service

[3]Dynamism means that the re-configuration of these federations upon service connections and disconnections is spontaneous

distributed on a local area network, and *iii*) service more widely distributed across several company networks.

In the rest of this sub-section a set of nomadic computing scenarios are described, in order to show that nomadic domains introduce new design challenges due to the heterogeneity of the involved technologies, the need for self-organization, the dynamics of the system composition, the application-driven nature, the co-operation with infrastructure-based networks, and the security hazards.

1.2.2 Networked Services: The Smart Airport

The user is willing to print a set of documents before departure; some documents are stored onto his Wi-Fi Personal Digital Assistant (PDA), whereas others can be retrieved through his company's document management web-service. As he switches the PDA on, its web browser is re-directed to the Airport's welcome page; upon an e-ticket based authentication, the system provides him with the list of the available services and facilities. The user notices that the system offers more powerful and dynamic services through a Jini-based service infrastructure. He can use these enhanced services, since his PDA is compliant with the Jini technology; thus, he browses the list of available printing facilities by means of his Jini-browser tool, and he localizes a Bluetooth laser color printer within the departure hall he is walking through; he notices that this printing service allows also to print remote web pages, as well as to print documents published on the world wide web as results of a specific Web-service. As the service is requested, the print-service's graphical client appears on the PDA. The user provides the tool with the local documents to print, as well as with the URI and service parameters to retrieve the remote documents by means of his company's Web-services. The tool provides him with an Airport Positioning System (APS),

which guides him to the printer; moreover, it informs him that the documents will be printed in five minutes. When the user reaches the printer to pick his documents up, an instant messaging service advertisement is received by his PDA. He joins this service, and subsequently contacts his friend Maggie, who is in the printer room at the moment. Maggie is printing her e-mails through her Bluetooth-printing enabled smart phone, which directly discovered the printer through its native Bluetooth interface.

1.2.3 Smart environments: A Digital Home Environment

User devices include mobile devices (ranging from powerful devices like notebooks and personal digital assistants to less powerful devices such as mobile phones), and fixed devices as well (e.g., printers, digital camcorders, and audio/video equipments). Mobile devices may create trusted and secure federations. The entire family can easily and conveniently watch TV and movies on any display, whether it is a TV or a PC, anywhere in the home, all at the same time. The program guide shows a common integrated listing of all movies, TV programs, pictures, music, and other content from all the devices on the home network at the touch of a button.

1.2.4 Ad-hoc Community: Interaction between Personal Digital Assistants

The future phone (or personal trusted device) will be the core of the personal networking system. It probes its surroundings looking for suitable peripheral devices such as displays, input devices, processors, fast access memories and access points to communication channels. It dynamically builds up the most appropriate end-user system that can be auto-configured. When two or more people having their future phones

gather together, they may want to create an ad-hoc community. The establishment and operability of the community cannot rely on availability of infrastructure services. Therefore, client functionality at the terminal is not enough to support applications usable for members of an ad-hoc community.

1.2.5 Personal Domain: Managing and connecting personal devices

A person can routinely use several devices for both professional and entertainment activities. The interaction between the user and his devices must be spontaneous as much as possible. When two personal devices interact (e.g., a bluetooth headset and a mobile phone) they may want to create an ad-hoc community. A Personal Area Network is established incorporating a number of devices so that the user can exploit his devices in the most natural and intuitive way.

1.3 Requirements of nomadic Environments

The presented scenarios explore various aspects of nomadic computing but they also show a common theme. The typical experience with nomadic computing is that of discovering usable services and of interacting with them while moving and upon movements. The Cross-Industry Working Team in the U.S. identified the following challenges in realizing a service infrastructure to support nomadicity [13].

Location-independent Communications. A significant challenge for nomadic computing is to make the process of establishing communications with a nomad largely transparent to both the nomad and his correspondent, regardless of which

one initiates the exchange. The service infrastructure must therefore supply an ubiquitous and reliable implementation of location independence which places no special burdens on the nomad.

Device Independence. The infrastructure must support user mobility by allowing for access devices with different capabilities, at different locations, through different access paths, and under different access conditions.

Widespread access. Because an essential component of nomadicity is that individuals remain as connected as they desire from wherever they desire, the communications devices and channels on which connectivity depends must allow for widespread access.

Security (identification, certification, and billing). Because the device is not the person, technology can hide the identities of the communicating parties. Ensuring that the individual at the other end of the communications channel is whom she says she is is critical to establishing effective access. Similarly, when the individual is in motion, switching between devices and locations, many other security issues are exacerbated.

Adaptability to new technologies. The rapid pace of technological change challenges the infrastructure to adapt continuously to new technologies. System components supporting nomadicity are likely to be among those changing most rapidly.

Heterogeneity of interaction platforms. In order to meet the requirements of each personal computing domain, a wide range of service discovery and interaction platforms have been proposed. While these platforms share a number of common attributes (e.g., the advertise-discover-interact paradigm), they each have distinguishing features. Since a comprehensive nomadic computing infrastructure should support all

the personal networking domains described in Section 1.2, future networked environments are likely to present developers with a heterogeneous environment composed of multiple specialized support platforms.

Partitioned communication environment A given person may use the same device (or set of devices) for accessing the infrastructure in various separate personal environments – business, family, holiday. The communications environment must be partitioned so that the user can filter and manage incoming communications.

Friendly interface. Public acceptance of new technology is largely based on the extent to which this technology is familiar and consistent with other technologies – that is, similar to models users already know. The complex interface devices and services that will probably characterize the infrastructure make realization of this friendly interface somewhat difficult. Nomadicity adds further complexity with its requirements for specialized devices and more complex connections.

1.4 Issues in realizing heterogeneous nomadic environments

The challenge for future nomadic computing systems will be the definition of an unifying infrastructure for the effective interworking of existing service platforms (rather than a brand new solution), and that fulfills the needs of personalized, context-aware, nomadic services and infrastructures at the same time [2, 4, 48].

This claim can be qualitatively confirmed upon an analysis of each domain's requirements, which is briefly summarized in Figure 1.1. For instance, applications within the **personal domain** are targeted to single users; they consist of device to device services. Spontaneous connectivity, device diversity, suitability to resource

	Personal Computing	**Ad-hoc community**	**Networked services**	**Smart-environments**
Location-independent communication	*N/A*	*Indifferent*	*Crucial*	*Desirable*
Device independence	*Crucial*	*Crucial*	*Desirable*	*Desirable*
Widespread access	*N/A*	*Desirable*	*Crucial*	*Desirable*
Security	*Desirable*	*Crucial*	*Crucial*	*Desirable*
Adaptability to new technologies	*Crucial*	*Desirable*	*Crucial*	*Crucial*
Heterogeneity of interaction platforms	*Crucial*	*Crucial*	*Crucial*	*Crucial*
Partitioned environment	*Desirable*	*Crucial*	*Crucial*	*Indifferent*
Friendly interface	*Crucial*	*Desirable*	*Desirable*	*Crucial*
Interaction	*1-1*	*1-N*	*N-N*	*1-N*
Involved devices	*<50*	*<1000*	*>100*	*<100*

Figure 1.1: The importance of requirements with respect to each nomadic computing domain. The evaluation refers to four levels of importance, namely *not applicable*, *indifferent*, *desirable*, and *crucial*.

(and power) constrained devices are the main challenges that the supporting infrastructures have to deal with. Hence, infrastructures for personal domain must take seriously into account heterogeneity and adaptability to new technologies, device independence, and friendly interface. Partitioning and security are not key challenges, as the number of devices is extremely small, and the involved devices usually belong to the same person.

As for the **ad-hoc community**, services require peer to peer messaging, collaborative applications, and advertising. Applications usually consist of device to device services, even though they can involve more devices than personal domain applications. Again, spontaneous connectivity, device diversity, suitability to resource (and power) constrained devices, are the main challenges the technologies have to deal with. This domain demands to partition the resulting environment more flexibly, as the number of devices involved in the discovery process is greater than the Personal Domain. Services require discovery and interaction between different persons. As infrastructure services are not available, the server functionality must be provided by the ad-hoc community. Since each peer in the distributed server needs to be light-weighted, the server functionality needs to be distributed among the members of the ad-hoc community according to a peer-to-peer model. Thus, the discovery and delivery processes are seriously concerned with security issues.

The **Networked services** domain encompasses Information and entertainment, M-commerce, corporate services, and advertising. This domain is characterized by a strong user mobility, which causes frequent changes of service environments; mobile users access services through portable computing devices (such as laptop and

handheld computers) in conjunction with both mobile and fixed communication infrastructures. Thus, it requires flexible mechanisms for locating and configuring very heterogeneous services provided either by mobile entities or by stationary hosts. Users wish to be able to locate the service easily and efficiently. Widespread access is the key challenge for networked services. Hence, the infrastructure requires spontaneous adaptation to the diverse resources of mobile terminals, to user preferences, and to other environmental parameters. Besides widespread access, the presence of both users and services, service composition, heterogeneity, and the partitioning of separate administrative domains are the main challenges that the service infrastructure has to face.

The **smart-places** domain requires seamless, dependable connectivity to the information users need to drive their professional life - i.e., digital office domain - or their personal life - i.e. digital home domain. The ability to bridge between both the digital office and the digital home is a key challenge for many applications in smart environments. The key requirements to address include adaptability to new technologies, high usability and friendliness of user interfaces, and the availability of several diverse platforms to support the discovery and the interaction of active devices (and services).

Since a global nomadic environment includes all the described nomadic computing domains, it is noteworthy to evaluate also the importance of each requirement with respect to the entire set of nomadic domains. Figure 1.2 shows the qualitative evaluation of the overall importance of the outlined requirements. This evaluation shows that, although each nomadic computing domain has its distinguishing requirements, the coexistence and rich heterogeneity of services and platforms is crucial for all the

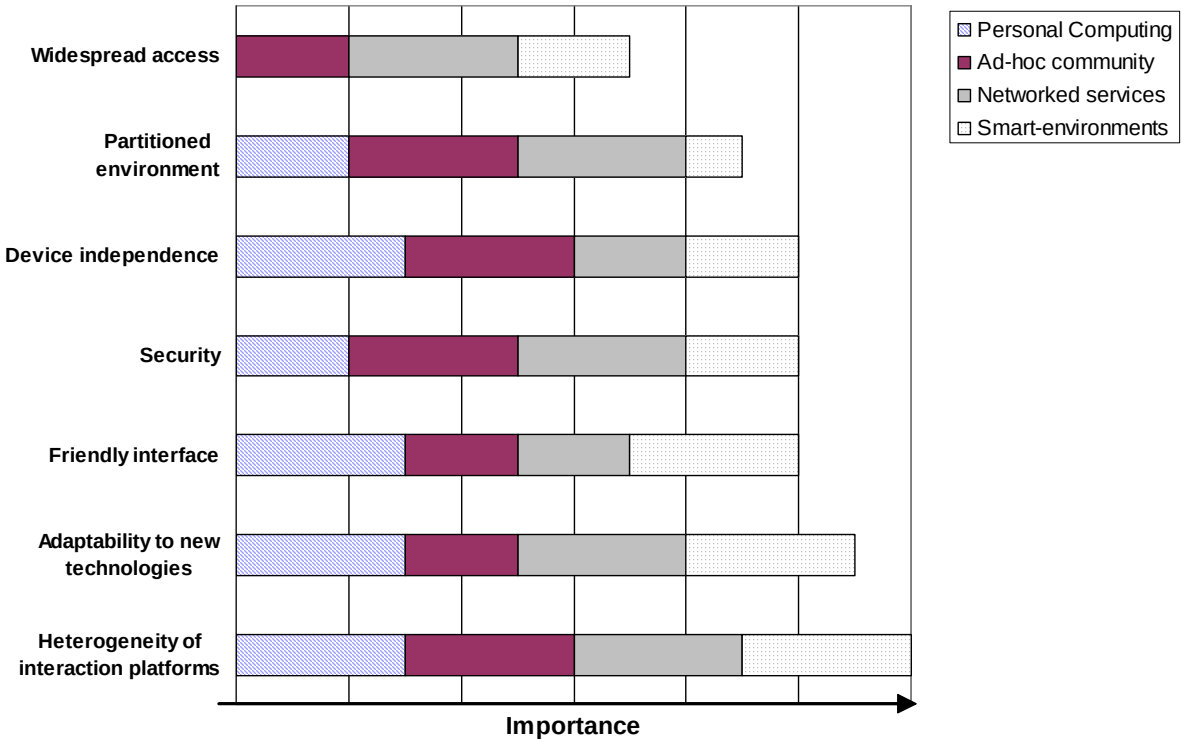

Figure 1.2: The overall importance of requirements compared to that of each nomadic computing domain.

envisioned scenarios.

Indeed, as the number of involved different platforms increases, a number of issues arise. Heterogeneity poses problems in terms of interoperability, scalability, context-awareness, and unpredictability. These problems are discussed in the remainder of this Section.

1.4.1 Interoperability issues

As for interoperability, a key issue is how to deal with the diversity of service representations and interaction models. For instance, certain platforms (e.g., UPnP and

WSA) use XML documents to represent services and define interactions, whereas others can define a specific URL syntax (e.g. SLP) or store service descriptors as serialized Java objects (e.g. Jini and RMI). Developing an application that is to interact with more than one of these platforms simultaneously poses considerable challenges to developers so far. Hence, an interworking mechanism for locating, interacting with and representing services is crucially required. As for networking technologies, the coexistence of diverse wireless networks can be challenging for they can interfere with each other. The interested reader may refer to [19] for further details about coexistence issues in wireless networks.

1.4.2 Scalability issues

As for scalability, even the performance of a single platform can significantly decrease as the number of clients and services increases. Indeed, discovery mechanisms are mostly based on multicast advertisements; as the number of clients and services grow, networks will be significantly impacted by service announcement traffic; thus system scalability can be affected by the burden due to dynamic discovery and interaction. Clearly, the coexistence of a huge number of heterogeneous services and platforms (which characterizes nomadic environments) exacerbates these issues, and it makes scalability be a prime consideration in the design of an interworking infrastructure. Hence, even with efficient protocols, further savings both in bandwidth and in dimensions of service registries – as well as consequent improvements in scalability – must be achieved.

1.4.3 Unpredictability issues

As for unpredictability, communication bandwidth and error rates change dynamically in wireless communication networks, a mobile system's battery power decreases, portable devices can be temporarily switched off or unreachable because of network partitions, and the monetary cost of communication can vary significantly. Envisaging a system that makes all these dynamics transparent is difficult. A practical solution could use specific intermediaries to handle the nomadicity concerns of unpredictable user behavior, unpredictable network, unpredictable computing, and graceful degradation. Developers have provided pragmatic solutions for these intermediary functions, but comprehensive general principles are missing. The issues to address encompass not only intermediaries' functionality, but also the integration of security, transactions, conversion overhead, and reliability. Moreover, although researchers have studied the viability of application adaptation in mobile systems, strategies for making adaptation decisions also require exploration.

1.4.4 Context-awareness issues

Context is defined by Dey and Abowd [52] as "any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant between the user and the application, including the user and application themselves". Context awareness is, then, defined as system's ability to "use context to provide relevant information and/or services to the user, where relevancy depends on the user's task".

As for context-awareness, nomadic applications should know their operating environment for context-dependent activities, such as giving directions or employing

more or less stringent security mechanisms. Hence, context-infrastructures should go beyond the traditional middleware approach: while in traditional middlewares, details are hidden from both users and application designers and are encapsulated inside the middleware itself (so that the distributed system appears to application developers as a single integrated computing facility), in nomadic environments it is neither always possible, nor desirable, to hide all the implementation details from the user [18], as applications may have valuable information that could enable the middleware to execute more efficiently and to behave according to user context.

Context-awareness introduces an additional layer of heterogeneity. Indeed, a crucial issue to address is how to make context-information interoperable. compatibility analysis. Due to the fuzzy, overlapping, and time changing nature of user contexts, context representation is often a crucial task. The uncertainty of context sources can be partially relieved by sharing information with neighboring devices that are involved in similar (i.e., matched) contexts. By collecting and sharing the context from other devices, one can achieve a more accurate description of the current context than the one recognized by single devices. However, context sharing exacerbates the heterogeneity of the resulting system, introducing novel differences between the involved devices.

1.5 Contribution

The aim of this thesis is to bring a significant contribution in this area, with the goal of enabling the definition of novel strategies to support interoperability between future nomadic users. The contribution of this work is to provide a comprehensive and thorough description of how to effectively deal with the heterogeneity of both

discovery and delivery infrastructures, and of specific applications as well.

As for the infrastructure, this thesis presents a novel architecture to address interoperability and scalability of both the discovery and the delivery of nomadic applications. It aims to mitigate the diversity of service representations, technologies, and interaction models of current nomadic computing domains. The proposed architecture allows to build a nomadic environment by composing nomadic computing domains (built upon current technologies), thereby allowing clients to discover and use services across domain borders. This architecture is composed of discovery and delivery agents: the former are in charge of connecting domains into a single logical domain, whilst the latter manage service sessions that users can establish to services after service discovery. The proposed solution uses a novel algorithm to decide which services to publish across diverse domains so as to reduce the burden due to service discovery across a huge number of diverse discovery platforms. This algorithm specifically uses both functional and technology-related constraints to drive the import and export operations. The proposed solution is evaluated both by simulation and by the implementation of a prototype used to connect Jini-based to Bluetooth-based domains.

As for specific applications, two case studies are thoroughly described so as to show how to deal with unpredictability and context-awareness. This work specifically shows how to handle the nomadicity concerns of i) location-awareness for mobile devices and of ii) unpredictable system behaviour for mobile multimedia services.

Chapter 2

Service Platforms for Nomadic Computing

This chapter lays the groundwork for the thesis and surveys the state of the art in service platforms for nomadic environments. A preliminary analysis of traditional and novel service platforms is provided in Section 2.1 and 2.2 respectively. Section 2.3 compares the suitability of the described platforms to satisfy the requirements of interoperability and adaptability to new technologies; it explicitly emphasizes the need for interworking in current and future nomadic computing environments. Section 2.4 concludes the chapter by describing and comparing on-going research works on interworking infrastructures for nomadic computing environments.

2.1 Traditional platforms

2.1.1 Jini

Jini technology - built on the top of the Java 2 Standard Edition (J2SE) platform - enables software services and devices to work together in an impromptu community [61]. Jini relies on Java's characteristics - such as code mobility, platform independence,

and distributed events - for reducing planning, installation, and human intervention in discovering and delivering services. The core of a Jini system is the so-called Lookup Service (LS); it allows clients to find and use services, and allows servers to join the Jini system. The LS provides service browsing, as well as white-pages (i.e. discovery by service interface), and yellow pages (i.e. discovery by service attribute). The yellow pages mechanism is based on exact matching: queries cannot include any relational operator. The presence of entities into the service directory is managed through registration leasing. The Lookup Service provides registered services with a lease token, in order to allow them to periodically renew their presence; in case of lease expiration, the corresponding service entry is canceled. LSs may be located by unicast and multicast discovery mechanisms. Jini also requires each device either to run a Java virtual machine or to associate itself with a device that can execute a JVM on its behalf.

Jini is certainly useful for the home environment, as long as interfaces for the desired devices are being developed. This is one of the drawbacks: Much is still left unspecified, interfaces for certain devices still have to be implemented, while they are already available for consumer electronics in other solutions. This prevents interworking between devices of different vendors. The Jini specification currently does not address security issues, even though latest specifications include some basic security mechanisms.

2.1.2 Salutation

Salutation [11] is an architecture for service discovery and delivery in ubiquitous computing environments characterized by strong mobility of user devices. Presence

and network-protocols independence are the main concepts in Salutation; both its discovery and delivery infrastructures are tightly related to such concepts. Salutation architecture is based on three main components, namely the Functional Unit (FU), the Salutation Manager (SLM), and the Transport Manager (TM).

A **Functional Unit** defines a service from a client's point of view. Functional units already specified or under consideration by the Salutation Consortium include printing, faxing, and document storage. Services can register and unregister functional units with the local Salutation Manager.

Salutation Managers function as service brokers. They are in charge of managing both the discovery and the delivery of services. They specifically *i*) manage service registry (to store service descriptors), *ii*) provide a discovery service, *iii*) monitor availability of registered services (through periodic heart-beat messages), and *iv*) manage service delivery sessions. As for discovery, Salutation Managers provide clients with three different mechanisms, namely exhaustive discovery of all available services (ALL CALL), discovery of a specific type of service (TYPE CALL), and discovery of the service that best suits several specified client characteristics (MATCH CALL). As for session management, the SLM establish a virtual communication pipe between client and service in order to send service invocations, results, and additional data. Communication on the virtual pipes is managed through the so-called Personality Protocols. A Salutation manager can operate in one of three "personalities". In *native personality*, Salutation Managers are used only for discovery. The *emulated personality* is similar to the native personality in that Salutation managers set up the connection, but in this case they transfer native data packets encapsulated in Salutation manager protocol format (bridge). In *Salutation personality*, Salutation

managers establish the connection between client and service, and they also mandate the specific format of the data transferred. The Salutation architecture defines the data formats.

Transport Managers isolate the implementation of the Salutation manager from particular transport-layer protocols and thereby gives Salutation network transport independence. To support a new network transport requires a new transport manager to be written, but does not require modifications to the Salutation manager. Transport managers locate the Salutation managers on their respective network segments via either multicast, static configuration, or reference to a centralized directory. Discovery of other Salutation managers allows a particular Salutation manager to determine which functional units have been registered and to allow clients access to these remote services. Communication between Salutation managers is based on remote procedure call (RPC).

A lightweight version of Salutation, namely Salutation-Lite, has been developed for resource-limited devices. It is based primarily on IrDA to leverage the large number of infrared-capable devices. Salutation-Lite focuses primarily on service discovery. The Salutation specification currently does not address security issues.

2.1.3 Bluetooth

Bluetooth (BT) wireless technology is a short-range communications system intended to replace the cables connecting portable and/or fixed electronic devices [5, 15]. It provides point-to-multipoint voice and data transfer. The technology supports both isochronous and asynchronous services. A simple isochronous application might link a cellular phone and wireless headset, where the headset and base are both Bluetooth

devices. More complicated applications include automatic discovery of wireless network connections and automatic synchronization of data between several Bluetooth devices.

Bluetooth protocol stack addresses communication between devices at several levels. [1] Groups of up to eight Bluetooth devices can form ad hoc networks called piconets to communicate, share services, and synchronize data. In each piconet, a master device coordinates the other Bluetooth devices (including setting the 1,600hopsper-second frequency-hopping pattern). Individual devices can participate in more than one piconet at a time.

The Bluetooth Service Discovery Protocol (SDP) provides a set of functionalities for enumerating the devices in range and browsing available services. It also supports stop rules that limit the duration of searches or the number of devices returned. Client applications use the API to search for available services either by service classes, which uniquely identify types of devices (such as printers or storage devices), or by matching attributes (such as a model number or supported protocol). Attributes that describe the services offered by a Bluetooth device are stored as a service record and are maintained by the device's SDP server. The distinction between service classes and descriptive attributes is not well defined, but service classes generally define broad device categories, such as Printer, ColorPrinter, and PostScriptPrinter, while attributes

[1] At the bottom of the stack, the radio and baseband layers provide the short-range, frequency-hopping radio platform. The link manager protocol (LMP) controls link setup and provides encryption and authentication services. The proprietary logical link control and adaptation protocol (L2CAP) provides multiplexed communication over LMP to higher level layers. L2CAP is proprietary, but other network protocols, such as IP, can be built on top of it. Specifically, higher layers can include the following protocols: Hayes compatible AT (ATtention) protocol, which provides a standard interface for controlling remote cellular phones and modems; RFComm, which emulates an RS-232 serial interface; a simple object exchange protocol (OBEX), which enhances Bluetooth's interoperability with IrDA; and Bluetooth's service discovery protocol (SDP).

allow a finer level of description. Manufacturers must eventually standardize these service classes for maximal interoperability between Bluetooth devices.

As for service delivery, SDP does not provide a mechanism for using discovered services, i.e., specific actions required to use a service must be provided by a higher level protocol. However, it does define a standard attribute *ProtocolDescriptorList*, which enumerates appropriate protocols for communicating with a service.

The key features are robustness, low power, and low cost. Moreover, many features of the core specification are optional, thereby allowing product differentiation.

2.1.4 Universal Plug'N'Play (UPNP)

The UPnP architecture defines common protocols and procedures to guarantee interoperability among network-enabled PCs, appliances, and wireless devices [16, 29]. It is a proposed architecture supported by the UPnP Forum, headed by Microsoft. UPnP aims to standardize the protocols used by devices to communicate by using XML messages. The basic building blocks of a UPnP network are devices, services and control points. Its device model is hierarchical. In a compound device, the root device is discoverable; a client (called a control point) can address the individual sub-devices independently. Devices that don't speak UPnP directly - called bridged devices - can be integrated into a UPnP network, thereby guaranteeing a basic support to interoperability. A bridge maps between UPnP and device-native protocols.

2.1.5 Universal Description, Discovery and Integration of business for the web (UDDI)

Universal Description, Discovery and Integration (UDDI) [10] is a specification for distributed Web-based information registries of Web services. UDDI registries rely on an information model defined in an XML schema. Such an XML-based approach allows to offer a platform-neutral view of data, and allows hierarchical relationships to be described in a natural way. The information provided in a UDDI business registration encompass: *i*) white pages including address, contact, and known identifiers; *ii*) yellow pages including industrial categorizations based on standard taxonomies; and *iii*) green pages, the technical information about services that are exposed by the business. Green pages include references to specifications for Web services, as well as support for pointers to various file and URL based discovery mechanisms if required.

2.1.6 Service Location Protocol (SLP)

The Service Location Protocol (SLP) [22] is an IETF protocol for automatic resource discovery and advertisement on local area IP networks. It is based on the interaction of three types of agent, namely the Directory Agent (DA), the Service Agent (SA), and the User Agent (UA). Directory Agents cache information about available services. Service Agents advertise the location and attributes of available services. User Agents discover the location and attributes of services needed by client software. User Agents can discover services by issuing a directory like query to the network. The SLP supports service browsing and string-based query for attributes, including relational operators and sub-string matching. SLP may also operate in a directory-less mode. Thus, it suits both ubiquitous and ad-hoc computing environments. SLP

also interacts with LDAP; hence services registered with an SLP DA can be automatically registered in an LDAP directory. This eliminates the need to reconfigure clients that already discover services using LDAP. Unlike Jini, Salutation, UDDI, and UPnP, which all aspire to some degree of transport-level independence, SLP is designed solely for IP-based networks.

2.1.7 Cooltown

Cooltown [30] is an infrastructure to support "web presence" for people, places and things, developed by HP Labs within the framework of the Cooltown project. Cooltown puts web servers into things like printers and puts information into web servers about things like artwork; it groups physically related things into places embodied in web servers. Using URLs for addressing, physical URL beaconing and sensing of URLs for discovery, and localized web servers for directories, Cooltown creates a location-aware but ubiquitous system to support nomadic users. On top of this infrastructure we can leverage Internet connectivity to support communications services. Web presence bridges the World Wide Web and the physical world, thus providing a model for supporting nomadic users without a central control point.

2.1.8 JXTA

The JXTA protocols are a set of six protocols that have been specifically designed for ad hoc, pervasive, and multi-hop peer-to-peer (P2P) network computing [59]. Using the JXTA protocols, peers can cooperate to form self-organized and self-configured peer groups independently of their positions in the network (edges, firewalls), and without the need of a centralized management infrastructure. From a high-level

point of view, JXTA can be divided into three layers: core, service, and application layer. The core layer provides fundamental functionalities like transport, discovery, and security primitives. The service layer relies on the core layer and provides services that are typically used in Peer-to-Peer applications like searching and indexing, file sharing, and security services. Applications usually rely on both, core and service layers, although the service layer may be more important to application developers.

2.2 Novel platforms

2.2.1 CARISMA and REMMOC: exploiting reflection

The first approach addresses service discovery issues at the middleware layer. This approach aims to manage diversity of mobile applications and services - which are implemented and advertised upon different service discovery and delivery protocols - through reflection and re-configurability [8].

2.2.2 Konark

The Konark middleware [9] specifically addresses discovery and delivery issues in ad-hoc networking. The main features of this architecture include automatic device configuration, peer-to-peer service discovery, service delivery, and repository distribution. Service delivery is based on the HTTP and SOAP protocols. Service discovery in Konark encompasses the so-called active-pull and service advertisement mechanisms: the former consists of a multicast request-reply mechanism initiated by clients, whereas the latter consists of a periodic advertisement sent by services in order to announce their presence to other entities (i.e. clients and other services).

2.2.3 Aura

Aura [56] is an architectural framework for ubiquitous computing applications. Aura represents an alternative approach that enables mobile users to make the most of ubiquitous computing environments, while shielding those users from managing heterogeneity and dynamic variability of capabilities and resources. Specifically, Aura has the following key features: first, user tasks become first class entities that are represented explicitly and autonomously from a specific environment. Second, user tasks are represented as coalitions of abstract services. Third, environments are equipped to self-monitor and renegotiate task support in the presence of run time variation of capabilities and resources. This architectural framework has a number of important benefits. By representing user tasks explicitly, Aura provides a placeholder to capture user intent. This knowledge is used to guide the search for suitable configurations in each new environment. By representing tasks as service coalitions, the infrastructure can recognize when all the essential services in a task can be supported, instantiating them jointly, or otherwise provide early warning to the user that that is not possible. By providing an abstract characterization of the services in a task, the infrastructure can search heterogeneous environments for appropriate matches to supply those services. By providing the environment with self-monitoring capabilities, the infrastructure can detect when task requirements (such as minimum response time) are not met, and search and deploy alternative configurations to support the task.

2.2.4 Sahara

The Service Architecture for Heterogeneous Access, Resources, and Applications (SAHARA) [49] aims to the composition of services and to seamless discovery. It is a technical architecture for application-level service composition and inter-operation across separate administrative domains. Its functional elements include service discovery, service-level agreements, and service composition under constraints. SAHARA supports peering and brokering, as well as diverse business, value-exchange, and access-control models.

2.2.5 Open Mobile Alliance and the Wireless Village

The Open Mobile Alliance (OMA) has published a set of enabler specifications to ensure the consistent development of mobile Internet Services [41]. It has identified service discovery as a key challenge for its Wireless Village (WV) Architecture; discovery of service in the WV is quite simple but interesting, as it relies on an authentication mechanism for establishing Service Level Agreements before discovering available services.

2.3 Comparing service platforms: the need for interworking

2.3.1 Service platforms and nomadic computing domains

The mentioned service platforms aim to raise the abstraction level of the user access to solve the problem of nomadic discovery and delivery: the trend is to discover services

not based on network addresses but based on service usage intentions. For service platforms to become nomadic, either a single technology must dominate or the most commonly used technologies must be made interoperable.

Indeed, each technology suits a specific personal computing domain. For instance, Bluetooth and UPNP seem to suit well the personal domain, as they provide an intuitive but powerful mechanism for device-to-device discovery and interaction. These technologies are suitable for a wide variety of mobile and fixed devices, such as mobile phones, home appliances, smart sensors, wearables, laptops, PDAs, and desktop PCs. The Jini technology is also suitable for this domain, even though its system requirements (i.e. presence of a Java 2 Standard Edition virtual machine on each device) do not match the capabilities of less powerful mobile devices (e.g., mobile phones, smart-phones, and PDA) and of low-cost devices as well. Moreover, service discovery in the personal domain should not rely on any underlying networking infrastructure, and on the presence of IP-based connectivity as well; this makes most of the cited technologies not suitable for the Personal Domain.

Bluetooth and UPNP seem to be suitable for the ad-hoc community domain too. They do not cope with the discovery of dynamically distributed peer-to-peer services. This issue is partially addressed by SLP and by the Konark middleware, since they can use a distributed registry for resource discovery. However, they pose several interoperability concerns, which could be solved by adopting an OSGi-like solution for managing device diversity (see next section).

As for the networked services domain, Salutation can provide for Presence and network-protocols independence, even though it does not manage service composition. SAHARA provides for service composition and inter-operation across separate

administrative domains. The Open Mobile Alliance (OMA) Instant Messaging and Presence Services (IMPS) specification is specially targeted to managing presence in interoperable end-user networked services.

2.3.2 The need for interworking

Since the requirements of each personal computing domain (that a nomadic computing system is composed of) are different from those of the others, the diversity of current technologies is the key factor to suit all the domains.

As an example, consider a wide-area nomadic environment composed of a huge number of smart environments each consisting of a federation of multiple devices. Such an environment demand that two different service discovery paradigms be adopted. Indeed, whilst most of device-oriented technologies (e.g., UPNP, Jini, Bluetooth) address the issue of the extreme dynamism of smart environments, these protocols do not scale to the wide-area. Rather, they are intended for use within relatively small communities of devices [2]. On the contrary, technologies for networked services (e.g., JXTA, UDDI) scale better to the wide-area, even though they do not provide the required spontaneity and dynamism. Hence, one should use at least two different technologies (e.g., UPNP and JXTA) in order to address the requirements both of inter-domain and of intra-domain discovery.

[2]For instance, the Windows ME UPnP client attempts to locate all the local network services upon initialization. If the multicast search yields n service advertisements then n TCP based HTTP interrogations follow to gather the XML service descriptions (the user would then initiate a further n interactions to get the presentation pages for each device). According to the UPnP specification, in the simple case of one root device with one service type, 12 packets are generated (4 packets each sent 3 times to avoid problems with packet loss) each refresh interval (normally 30 minutes). As a client joins or roams into the network, it issues an *SSDP:ALL* M-SEARCH request (3 times for reliability). Every service must respond to each M-SEARCH by unicasting its service advertisement to the client (again 3 times per response). A single search yields a total of 36 response datagrams (if no packets are lost).

This simple example shows that interworking seems to be the most promising prospect for interoperability in nomadic computing: it is very unlikely that either *i*) one technology will fit all the requirements of a nomadic environment or *ii*) device manufacturers will embrace multiple service discovery and delivery technologies on mobile devices. This is especially true with respect to low-cost and embedded devices, i.e., with respect to personal devices used by nomadic users. Hence interworking efforts are perhaps the most important force in nomadic computing systems deployment.

However, though the current service platforms succeed in supporting a particular nomadic computing domain, they have a limited support for very heterogeneous environments. Figure 2.1 shows a comparison of the presented platforms with respect to the issue of realizing a very heterogeneous nomadic environment. The main limitation of current technologies is the limited support for interoperability which in turn limits also their adaptability to new technologies [50]. A few of them can support interoperability by applying ad-hoc bridging strategies rather than a more open and general approach [21]. As for scalability, SLP and Konark support registry distribution, whereas Salutation and Sahara try to dominate the complexity of the resulting system through a hierarchical organization of discovery and delivery agents. Task distribution in Aura can also be used to leverage the scalability of its delivery infrastructure. It is worth noting that the less dynamic and self-configuring the platform is, the better the resulting infrastructure scales to large systems. The second and third columns in Figure 2.1 show that Konark is the only platform which tries to address both scalability and unpredictability issues, whereas the rest of the platforms cope with either the former or the latter issue. Besides reconfiguration and user mobility,

	Support for Interoperability	Support for scalability	Support for unpredictability	Support for Context-awareness
JINI	*No*	*No*	*Yes (self-reconfiguration)*	*May support (discovery)*
Salutation	*Yes (bridging)*	*Yes*	*May support*	*No*
Bluetooth	*No*	*No*	*Yes (channel and radio layers)*	*May support*
UPNP	*Yes (bridging)*	*No*	*May support*	*No*
UDDI / SOAP	*Yes (XML)*	*Yes*	*May support (mobility)*	*No*
SLP	*Yes (SLP – LDAP integration)*	*Yes (centralized or distributed registry)*	*No*	*No*
Cooltown	*No*	*Yes*	*No*	*May support (sensing and location-awareness)*
JXTA	*No*	*Yes*	*May support*	*May support*
CARISMA	*No*	*no*	*Yes*	*Yes*
Konark	*Yes*	*Yes (distributed registry)*	*Yes (automatic reconfiguration)*	*May support*
Aura	*No*	*May support (task distribution)*	*Yes*	*Yes*
Sahara	*Yes*	*Yes*	*No*	*No*
OMA	*May support*	*No*	*May support (mobility)*	*No*

Figure 2.1: A comparison of the surveyed technologies

CARISMA and Aura are the only platforms which support variability of capabilities and resources. They also provide an explicit support for context-awareness, whereas almost all of the presented platforms do not support it.

2.4 Interworking solutions

For service platforms to become ubiquitous, either a single platform must dominate or the most commonly used technologies must be made interoperable. Currently,

bridging seems to be the most promising prospect for interoperability [50]. The need for interworking is also present in certain recent research works on supporting nomadic computing. The IST-NOMAD project [14] deals with the integration and composition of location-aware service discovery mechanisms, handover procedures and service / user profiling, by developing technology that allows users to freely roam across existing and future network infrastructures.

The OSGi Alliance has recently released the third release of the OSGi Service Platform [28], an enhanced software specification for the delivery of managed services to devices in homes, cars, businesses, and other environments. The specification is explicitly designed to be open and synergistic with a wide range of existing networking and computer technologies. The specification deliberately does not prescribe any particular device or network technology. Nor does it specify a particular device discovery method. Rather, it focuses on the attachment of devices supplied by different vendors. OSGi provides a low-level mechanism for device discovery which can be used as a building block for more complex discovery architectures, as demonstrated by the work in [39].

As for XML registries, the Java API for XML Registries (JAXR) provides a uniform and standard Java API for accessing different kinds of XML Registries. An XML registry is an enabling infrastructure for building, deploying, and discovering Web services. Currently there are a variety of specifications for XML registries including, most notably, the ebXML Registry and Repository standard, which is being developed by OASIS and U.N./CEFACT, and the UDDI specification, which is being developed by a vendor consortium. JAXR enables Java software programmers to use a single, easy-to-use abstraction API to access a variety of XML registries. Simplicity

and ease of use are facilitated within JAXR by a unified JAXR information model, which describes content and metadata within XML registries. As an abstraction-based API, JAXR gives developers the ability to write registry client programs that are portable across different target registries. Similarly, JAXR also enables value-added capabilities beyond those of the underlying registries. The current version of the JAXR specification includes detailed bindings between the JAXR information model and both the ebXML Registry and the UDDI Registry v2.0 specifications.

As for discovery, the integration between the discovery infrastructures of different nomadic domains has been addressed by using discovery brokers (ie., discovery agents). Such brokers can be implemented according to a service query translation or service registration translation approach. Service query translation is about sending service queries across service discovery domain borders, whereas service registration translation is about sending (translated) service registrations across service discovery domain borders [32]. The Salutation architecture [11] is based on a broker. Salutation has some significant differences compared to the proposed architectural framework: *i*) Salutation's service broker is more closely coupled to applications; *ii*) Salutation uses service query translation approach rather than service-registration translation; and *iii*) Salutation defines a well-defined translation between service ontologies. Conversely, the solution proposed by this thesis is presented as an architectural design pattern for discovery brokers, thereby leveraging the re-usability of the entire approach. A Jini-SLP bridge has also been developed, which allows services lacking a JVM to participate in Jini systems [21]. The heart of the Jini-SLP bridge is a special SLP User Agent that acts as discovery broker and registers the availability of "Jini-capable" SLP Service Agents. To do this, Jini-capable SLP services advertise

the availability of a Jini driver factory. When a Jini client needs one of the registered SAs, it downloads a Jini-driver factory from the lookup server and uses it to instantiate a Java object to drive the service. Similar schemes are possible for the other technologies; for example, it should be possible to Jini-enable UPnP services in this way. Miller and Pascoe describe mapping Salutation to Bluetooth SDP to take advantage of Bluetooth's wireless capability [38]. Two approaches are considered: the first maps the Salutation APIs to Bluetooth SDP by implementing Salutation on top of Bluetooth; the second uses a Bluetooth transport manager and essentially replaces Bluetooth SDP with Salutation. This approach will also work with other schemes, like Jini. Bluetooth is a particularly attractive target for interoperability, primarily because of its wireless capability. Because of this, additional interoperability efforts between Bluetooth and other service discovery technologies seem inevitable.

Although the above mentioned bridging strategies allow to leverage the interoperability of certain existing interaction platform, the main limitation of such strategies lies in the lack of effective advertisement-filtering mechanisms which makes them unsuitable for large-scale environments. Futhermore, as the amount of domain increases, the amount of needed mappings grows quadratically. Instead, the existence of a general domain (internal to the service brokers) enables implementing only the mappings between the general templates and the (external) service templates. Hence, the proposed solution allows the growth of the amount of mappings to be limited to linear by the general templates. Moreover, the mentioned strategies show how to connect two (or more) service discovery domains establishing a technology-to-technology bridge, thereby providing an individual solution to specific integration problems. Conversely, this work proposes a re-usable architectural pattern for building a comprehensive

interworking infrastructure with a technology-independent strategy.

As for integration, a more versatile discovery-broker approach has been proposed by Koponen and Virtanen in [32]. The work by Koponen and Virtanen bears a resemblance to this thesis, for it is based on a general domain for service-registration translation to address the need for federating service discovery domains. However, the work in [32] focuses mostly on simple local federations; wide area service discovery is omitted completely, whereas this thesis proposes an interworking infrastructure which addresses scalability to wide-area federations as well. Moreover, the architecture proposed by Koponen and Virtanen focuses on mediating the data (service registrations) and not on mediating service requests, thus ignoring service-delivery issues. Conversely, the architectural framework herewith proposed can be used to manage both service advertisements and service sessions.

As for interoperable context-awareness, there is an increasing interest in defining a high-level software application programming interface for technology-independent location-awareness [43]. Several organizations have recently proposed location APIs to leverage the interoperability of both indoor and outdoor positioning systems. In order to deal with the heterogeneity of location techniques, the Java Community Process (JCP) has recently finalized a Java Specification Request (JSR) to define a Location API (JSR-179) [47] for MIDP-compliant devices. This package provides applications with functionality for *i*) obtaining information about location and orientation of the mobile device, and for *ii*) accessing a shared database of known locations (the so-called Landmarks). These specifications may be implemented by means of existing location methods, including satellite based methods like GPS, as well as short-range positioning methods. The Open Mobile Alliance (OMA) Location Working Group

(LOC), which continues the work originated in the former Location Interoperability Forum (LIF) [42], is developing specifications to ensure interoperability of Mobile Location Services on an end-to-end basis. This working group has defined a location services solution, which allows users and applications to obtain location information from the wireless networks independently of the air interface and positioning method. The Finnish Federation for Communications and Teleinformatics (FiCom) has built a national location API to enable location-information exchange between network operators, and between operators and service providers as well [20]. FICOM's API is compliant with the LIF location services specifications.

Chapter 3

The Proposed Architectural Framework

3.1 Conceptual model

Section 1.4 showed that four main problems stem from the heterogeneity of nomadic environments, namely interoperability, scalability, unpredictability, and context awareness. This chapter proposes a novel architectural framework for nomadic computing systems which is specially designed to overcome these issues.

Interoperability and scalability issues are mostly related to the characteristics of each interaction platform, whereas unpredictability and context-awareness issues depend on application-specific requirements. Driven by these differences, the framework combines application-level and platform-level strategies in order to fulfill heterogeneity requirements comprehensively.

The conceptual model of the proposed framework is strongly influenced by such a combined approach, as Figure 3.1 shows. The main idea behind the framework is to add interoperability and scalability to existing interaction platforms by means of

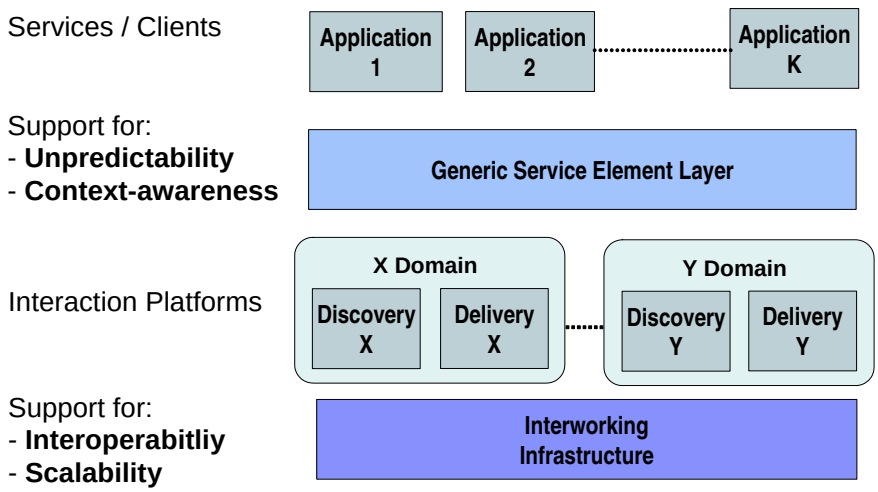

Figure 3.1: Conceptual model of the proposed architectural framework

a novel interworking infrastructure. The framework must thereby define placeholders for capturing service advertisements and service delivery sessions, and for representing the nomadic computing environment as the composition of several service-provisioning domains as well.

As for the heterogeneity of context-information and for the unpredictability of system behaviour, the framework provides a number of basic services or service-elements to applications. Therefore, the framework must also define the major abstractions to address heterogeneity from this application-dependent point of view.

Such abstractions are represented in Figure 3.1 as a toolbox of Generic Service-elements on which context-aware and adaptive applications can rely. In order to leverage the re-usability of such service-elements, it is crucial that a set of interfaces be defined to represent the functionality required by nomadic applications. It is worth noting that the adopted approach is quite similar to that of middleware platforms with respect to traditional distributed systems. Middleware is a widely used term

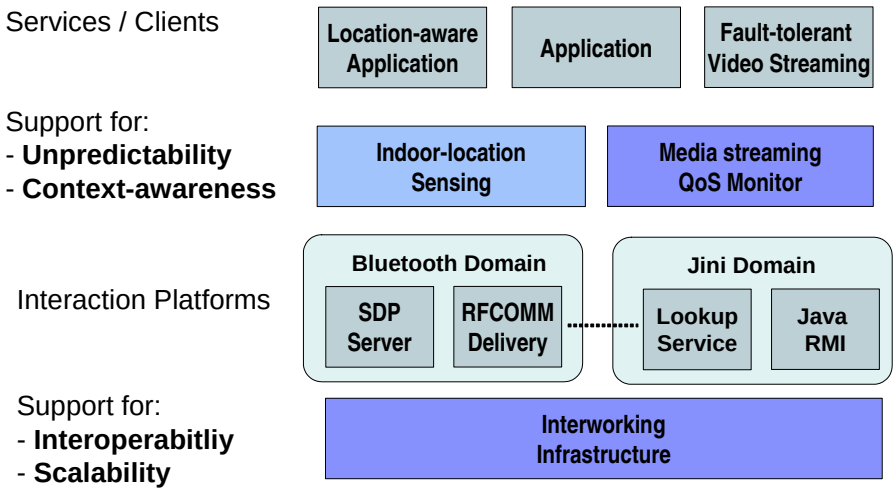

Figure 3.2: An architecture resulting from the proposed architectural framework

to denote a set of generic services above the operating system. The importance of middleware as a set of generic services above the operating system and transport stack is widely recognized. Such a modular approach is also used in the herewith presented architectural framework to address context-awareness and unpredictability. Chapter 5 will show how to design and to implement two service-elements each supporting a well-known class of applications.

Figure 3.2 shows a bird's eye view of an example system resulting from the proposed architectural framework. More specifically, it shows that the interworking infrastructure can be used to federate Jini and Bluetooth domains, thereby adding interoperability to both the interaction platforms. It is worth noting that the infrastructure can also widen the boundaries of a bluetooth service, allowing even clients located on the wide-area to discover and use it. Figure 3.2 clarifies also the role of

generic service-elements within the proposed framework. For instance, as for unpredictability and context-awareness, location-aware applications could further exploit location-sensing elements to build specific location-attributes and models, whereas multimedia applications could exploit media-streaming-qos monitoring elements to adapt the behaviour of their encoders and decoders accordingly.

3.2 The Interworking Infrastructure

3.2.1 The overall architecture

The interworking infrastructure allows to build an interoperable nomadic environment by composing several nomadic computing domains. Such an infrastructure relies on a group of cooperating agents which address interoperability and scalability issues from different points of view. The combined goal of these agents is to enable service discovery and delivery across domain borders, thus leveraging the interoperability of the involved domains by merging them into a global service-provisioning environment. However, when dealing with environments composed of a huge number of domains, it is necessary to take both scalability and partitioning issues into account, as mentioned in Section 1.4.2. To this aim, the global environment is assumed to be hierarchically structured. Its hierarchical organization relies on the concepts of **Nomadic Computing Domain**, **Domain-Federation**, and **Global Environment**. **Nomadic computing domains** are the elementary service-provisioning domains and represent a single physical end-system composed of devices that use a single interaction platforms to discover and deliver services. A **Domain-Federation** is a wider service provisioning environment which can include several Nomadic Computing Domains.

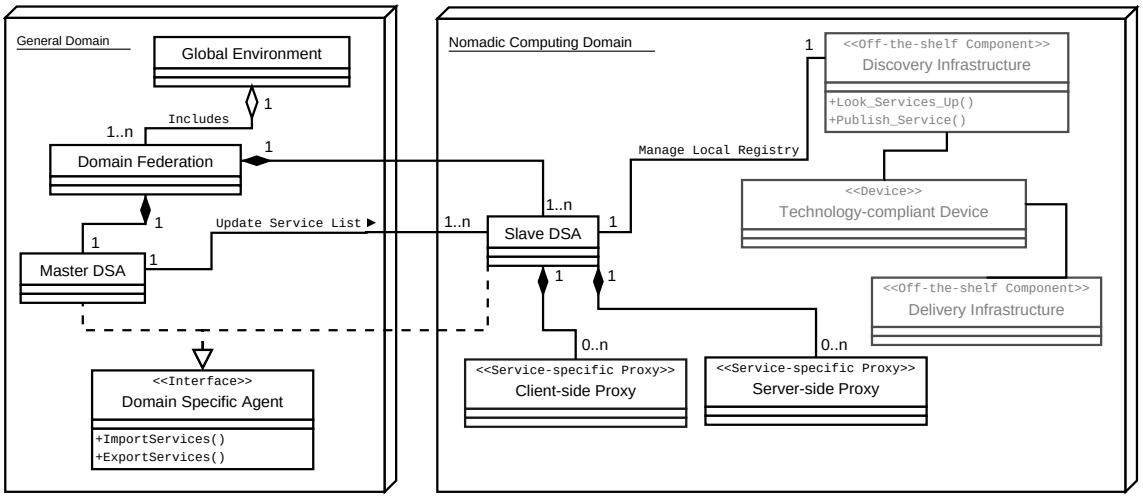

Figure 3.3: The conceptual model of the interworking infrastructure. Conceptual-classes in grey represent off-the-shelf components.

The aggregation of several Domain-Federations results in the creation of the **Global** (nomadic computing) **Environment**.

The conceptual model of the proposed interworking infrastructure is depicted in Figure 3.3. The infrastructure is composed of the following components. First, the discovery agents, called **Domain Specifc Agents** (DSAs), are in charge of connecting diverse domains into a single logical domain. Second, the delivery agents, called **Service Specific Proxies**(SSPs), manage service sessions that users can establish to services after service discovery. More specifically, Service-Specific Proxies are software components that provide a mechanism for services to be adapted and delivered even to incompatible devices.

From a logical standpoint, a domain comprises one instance of Domain Specific Agent and several instances of Service Specific Proxy. However, in order to suit the hierarchical organization of the global environment, the framework defines two hierarchically ordered types of discovery agents, called **Master DSA** and **Slave DSA**.

Each federation is assigned a specific Master DSA, whereas a Slave DSA interfaces a single domain to a certain federation.

3.2.2 Managing and updating service lists

Each agent holds a list of all the services published within its own service-provisioning environment. A Master DSA's service-list contains the list of all the services currently published into the federation of its own (**federation-level service list**), whereas that of a Slave DSA comprises all the services currently published within the nomadic computing domain of its own (**domain-level service list**). Such lists consist in a set of descriptive documents, called **Interoperable Service-Description Records** (ISDRs), each assigned to a specific service. Section 4.1 will provide further details about the data comprised in such documents, and about their structure as well.

The integration of different interaction platforms can be achieved according to a service query translation or a service description record translation approach. The former is about sending service queries across domain borders, whereas the latter is about sending translated service descriptions. The proposed infrastructure adopts a service description translation approach for the following reasons. First, as the infrastructure aims to transparently integrate diverse domains, description translation is more appealing than query translation; indeed the set of query and notification functionalities of current service discovery solutions is more heterogeneous and more dynamic than the set of current service description templates. Second, changes in service queries are more frequent on network than changes in service descriptions. Third, as the infrastructure must enable service delivery across a domain's boundaries, the description translation approach is more suitable for the dynamic creation

of Service-Specific Proxies both on the client-side and on the server-side. Finally, the presence of Master DSAs allows for grouping multiple service descriptors into a single coarse-grained advertisement. This federation-level advertisement is crucial to make the infrastructure scale to a large number of nomadic domains, and to screen a certain federation from undesirable and unauthorized direct access to/from slave agents residing in different federations.

The Slave DSA is in charge of discovering services within a single nomadic computing domain, and of providing its Master DSA with the domain-level service list. On the one hand the Master DSA is responsible for the propagation of service description records among the various SDSAs of its own federation, while on the other hand it is in charge of providing other federations with the service list of its own federations. That is to say, the role of a Master DSA is two-fold as it provides key discovery services at the Federation / Global Environment interface (*inter-federation discovery*), and within its own Federation as well (*intra-federation discovery*).

Service-description translation requires the framework to define a mechanism to trigger translations and exchange of information between the agents involved in intra-federation and inter-federation discovery. More specifically, the service lists could be updated by using either a reactive or a proactive approach. In the reactive approach, the owner of a service-list can decide to update it upon specific events. In the proactive approach, the agents take action by firing events and they do not only react to events when they happen. That is to say, such updates are triggered by third-parties (another agent) which can force a certain agent to update the service list of its own.

As for **inter-federation discovery**, the MDSA provides SDSAs with a mechanism to retrieve the list of all the services available in the global environment. To

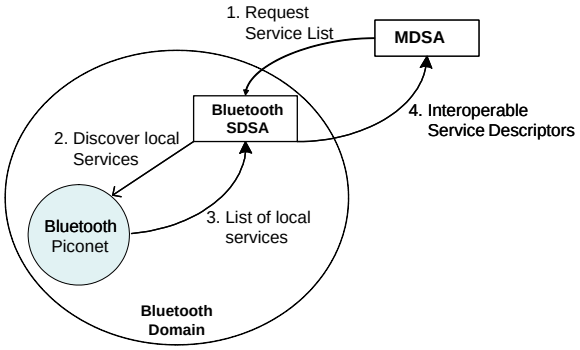

Figure 3.4: Exporting service advertisements

this aim it retrieves the service-lists of each external federation by contacting all the MDSAs comprised in the nomadic environment. As for **intra-federation discovery**, the Master DSA provides each SDSA within its own federation with the list of the services available in the rest of the Federation. To this aim it firstly retrieves the local-service-lists of each domain within its own federation by contacting all the comprised SDSAs, and subsequently sends the federation-level service list to all the Slave DSAs within the Federation.

Inter-federation discovery is reactive, for it is must be explicitly triggered by the interested Master agent upon a specific event. Conversely, intra-federation discovery is proactive, since the Slave DSA updates its own domain-level service list upon the arrival of its own MDSA's requests. More specifically, the proactive intra-federation discovery mechanism comprises the following steps. During the first step, depicted in Figure 3.4, it retrieves Service Description Records (SDRs) from the local service registry (e.g., a set of Bluetooth Service Discovery Protocol servers, or a Jini Lookup Service) and converts them to an Interoperable Service Description Record format which can be understood by other DSAs. Subsequently, it sends a set of ISDRs to its

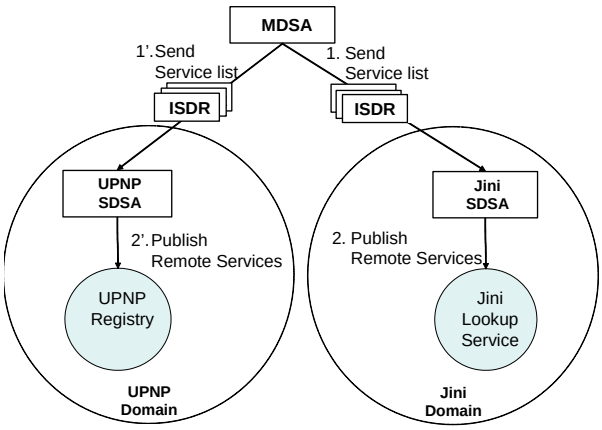

Figure 3.5: Importing service advertisements

Master DSA, which can thus gather the incoming ISDRs in order to build the list of all the available services within its federation. This list is then autonomously forwarded by the MDSA to its Slave DSAs, as shown in Figure 3.5. Upon the arrival of such service advertisements, a Slave DSA translates them from the common semantics and syntax of the ISDRs to those of its own domain (e.g., those of Bluetooth descriptors). Such translated descriptors are subsequently published by the SDSA into the domain of its own, thus providing placeholders for external services to be discovered as local-domain services.

The description of MDSA responsibilities given above has shown also that the slave DSAs are in charge of importing and exporting services within the domain of their own. However, SDSAs are involved not only in service-advertising (i.e. supporting the interoperability of service discovery infrastructures) but also in supporting the interoperability of service delivery infrastructures. Hence, the import/export operations can also require SDSAs to create/destroy service-specific proxies.

Chapter 4

Addressing Interoperability: the Interworking Infrastructure

This chapter provides further details about the design and the implementation of Master and Slave *Domain-Specific Agent* components. The chapter also includes a thorough description of a novel algorithm for discovery agents to mitigate the diversity of service representations and interaction platforms. This algorithm specifically uses both functional and technology-related constraints to drive import and export operations. The rest of the chapter is organised as follows. Section 4.1 introduces preliminary definitions and concepts. Design guidelines for the internal structure of discovery agents are provided in Section 4.2. Section 4.3 introduces the novel filtering algorithm to import and export services across heterogeneous domains. The stability of the proposed algorithm is evaluated by simulation. Section 4.4 describes the implementation of a system prototype, which is composed of Bluetooth-based and Jini-based Domain Specific Agents. The agents have been tested in a real environment in order to show how to add interoperability to an existing service provisioning domain by means of the developed infrastructure. Such a proof-of-concept implementation is described in Section 4.5.

4.1 Preliminary definitions

As defined in Section 3.2, the Interoperable Service Description Record (ISDR) is used to translate service advertisements between heterogeneous domains. An ISDR comprises the following sections. The *Basic Information* section contains the service identifier, as well as a set of description attributes (e.g., service name and a set of keywords). The *Service Profile* section defines service semantics in terms of well-defined service classes (e.g., printing service, web-connectivity, instant messaging). The *Client Resource* section contains optional information about system requirements on the client-side (e.g., required amount of RAM and storage space, required operating systems, and supported wireless technologies). The *Service Access* section provides details about the platform used to deliver a certain service. It specifically provides information about the adopted technology, as well as a technology-specific identifier which allows clients to establish a service session so as to start service provision.

In order to decide whether a descriptor can be translated or not, it is necessary to define the concepts of *descriptor translatability* and *service visibility*.

A generic Service Description Record SDR_k is a list of attributes $SDR_k = (a_1, a_2, ...a_n)$. Each field a_j is assigned a name, a value, and specific semantics (e.g. the *name* field may be assigned a *printService* value, and "name of service" semantics). Fields can be either mandatory or optional. It is noteworthy that the cardinality and the optional nature of SDRs depends on the specific domain that they refer to.

Let D_1 and D_2 be two different domains. Given two different service descriptors $SDR_1 = (a_1, a_2, ...a_n) \in D_1$, and $SDR_2 = (b_1, b_2, ...b_m) \in D_2$, the following properties can be defined:

Descriptor Translatability: a descriptor $SDR_1 \in D_1$ is *translatable* into a

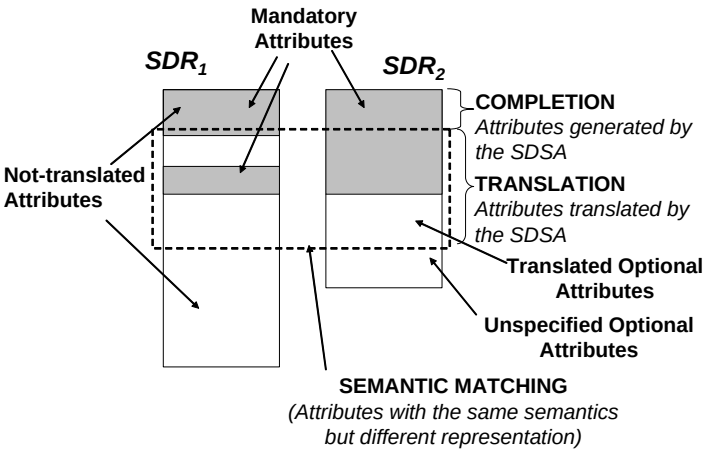

Figure 4.1: Semantic matching, translation, and completion of Service Description Records

descriptor $SDR_2 \in D_2$, being $D_1 \neq D_2$, if the following conditions hold: *i*) all information representing the same semantics (**semantic matching**) can be translated (**translation**), and *ii*) the remaining mandatory fields of the target SDR can be calculated by processing either a subset of SDR_2 fields or an external piece of information (**completion**). Figure 4.1 shows the role of completion, semantic matching, and translation with respect to descriptor translatability.

Service Visibility: a service $s \in D_1$ is *visible* in domain $D_2 \neq D_1 \Leftrightarrow \exists SDR_2 \in D_2 \mid SDR_1$ is translatable in SDR_2. It is worth noting that if a service $s \in X$ is visible in domain Y, and its Y-version is visible in another domain Z, then s is visible in Z. If such a condition holds, s is said to be *visible in Z trough Y* (**transitive-visibility property**).

The overall integration approach relies on this *Transitive-visibility* property. Indeed, this property allows Domain Specific Agents to integrate Nomadic Computing

Domains by using the Global Environment as an intermediary for service advertisement and for discovery as well.

Advertisement Filtering: in large-scale nomadic computing systems, discovery agents can generate a large volume of service advertisements. However, each domain may be interested in only a small portion of such advertisements. For instance, a certain domain could be interested only in the advertisements that concern a specific set of service types (e.g., messaging and network access). Moreover, visibility cannot be guaranteed for all the services to import. Hence, large-scale interoperable systems need directory services that aggregate service advertisements, manage queries from clients, and decide which services to publish across diverse domains. Such an import mechanism is called **Advertisement Filtering**; it can be driven by different principles, such as resource (i.e., bandwidth, registry dimension) saving strategies, description semantic, and functional and technological constraints.

Filtering Stability: a filtering mechanism is stable if the average amount of imported services is constantly smaller than the overall number of services in the registry.

When designing a filtering mechanism, it is crucial that the following two requirements be taken into account. First, the mechanism should allow the discovery agent to control the distribution of different types of service (e.g., printing, messaging, media-streaming) within its child domain, and to behave accordingly as well. Second, it should allow the agents to tune the stability of the resulting filtering mechanism.

4.2 Designing Domain Specific Agents

4.2.1 Designing a Slave Domain Specific Agent

According to Section 3.2, the major responsibilities of a Slave DSA comprise the following tasks:

import/export : to import and to export services within a single nomadic computing domain;

filtering : to filter the received service-list upon import operation, according to visibility rules as defined in Section 4.1;

translation : to translate Interoperable SDRs into local technology-specific description records (and vice versa);

proxy management : to create and destroy service-specific proxies;

external advertisement : to provide a mechanism for Master agents to discover the services of a certain domain within the federation of their own.

Based on the outlined responsibilities, a Slave DSA has been designed as resulting from the interaction of several components. The internal architecture of such a slave agent is presented in Figure 4.2 and briefly described in the following. It is worth noting that the Slave DSA is directly connected to a specific interaction platform. Hence, its implementation is highly dependent on the adopted target platform, as Section 4.4.1 will show.

The **Publishing Manager** is the key component for import and export operations. It creates and maintains the ISDR-based registry of local services. More

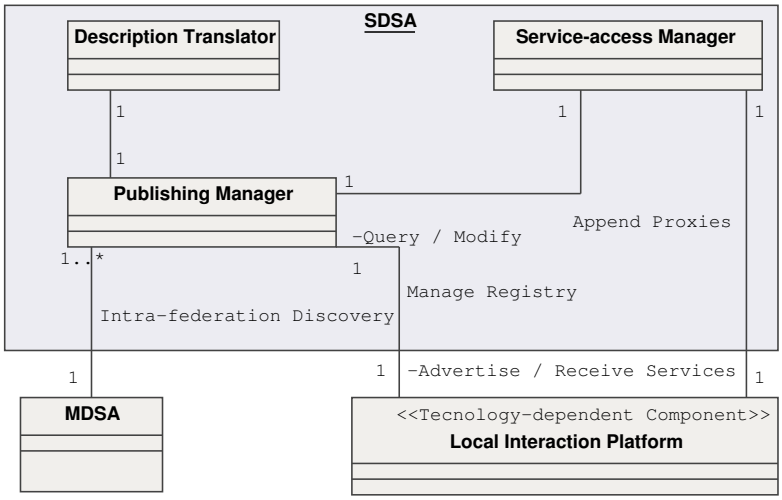

Figure 4.2: High-level UML Class Diagram of the Slave DSA component

specifically it is in charge of *i*) filtering the service list before importing services, *ii*) importing and discovering services within the domain associated to the Slave DSA, and *iii*) coordinating other components in order to publish external services into the local domain and to advertise the local services externally.

The **Description Translator** is in charge of translating service descriptors. Such a component evaluates the translatability property to decide whether a service is visible or not; hence, it is responsible not only for translation but also for completion.

The **Service Access Manager** is responsible for the creation and for the destruction of service specific proxies. It is also in charge of producing information for creating the *Service Access* section of the Interoperable SDR.

Importing and exporting service advertisements

Figure 4.3 shows that the internal structure of the Publishing Manager component comprises the following classes.

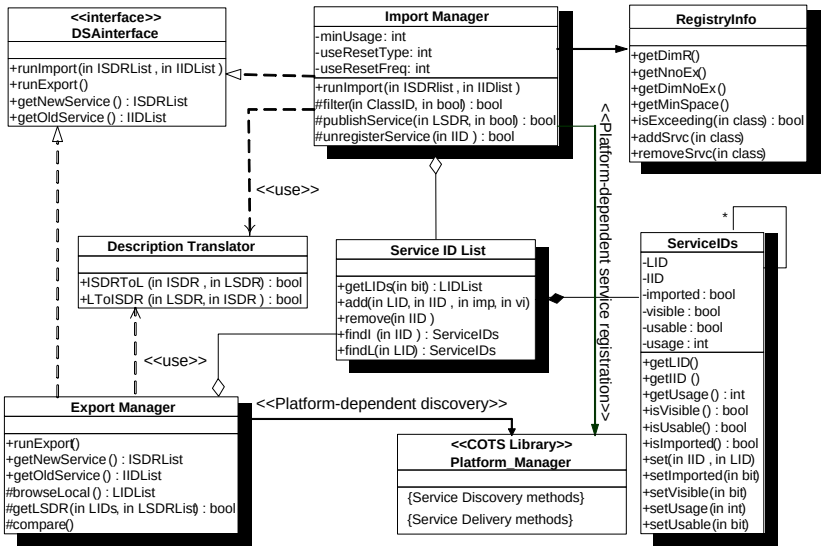

Figure 4.3: UML Class Diagram of the major classes involved in the management of service advertisements. Shadowed classes belong to the Publishing Manager.

The **Import Manager** is in charge of registering external services onto the local domain. Similarly the **Export Manager** is responsible for the announcement of local services to external domains. As for the specific domain to integrate, both the Import and the Export Manager use the **Platform Manager** component to query and/or to modify the local domain. The Platform Manager class is the interface to the discovery and delivery functions of the adopted interaction platform. Such key-functional blocks are typically provided as a set of off-the-shelf components.

Advertisement filtering and semantic matching

In order to investigate the translatability of a certain service and to translate its service descriptor, the Import Manager and the Export manager components use the Description Translator. The internal architecture of the Description Translator is

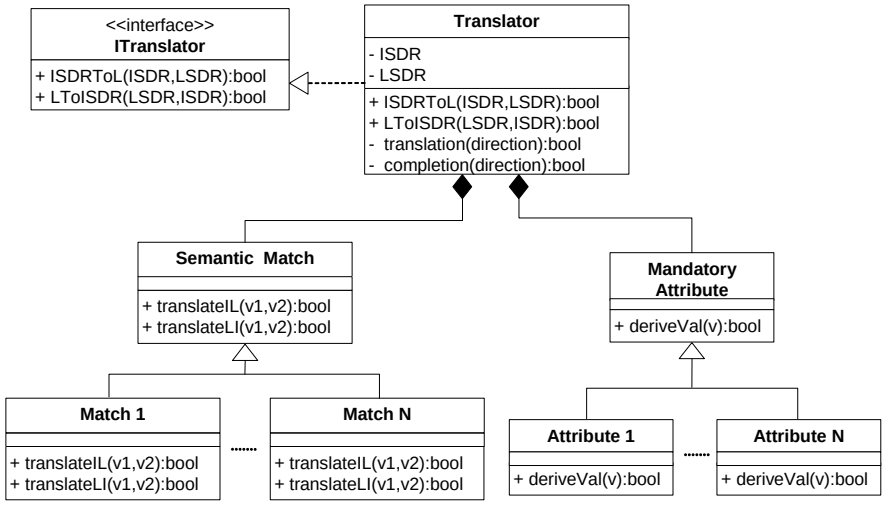

Figure 4.4: UML Class Diagram of the Description Translator component

depicted in Figure 4.4. Such a class diagram represents the conceptual architecture of a Description Translator component. Indeed, the number and the signature of all the member functions in Figure 4.4 is highly dependent on the adopted interaction platforms. The Translator class is responsible for establishing the translatability of a certain service. To this aim, it exploits both the Semantic Match and the Mandatory Attribute classes. The former is used to translate both mandatory and optional translatable attributes, while the latter is used to calculate the values of those ISDR attributes that are mandatory but not obtainable by the Local (platform-dependent) SDR (completion). Even though the Import Manager and the Export Manager exploit the same translation mechanism, they adopt different advertisement-filtering strategies. While on the one hand the Export Manager is only in charge of discovering and of trying to translate all the local services, on the other hand the Import Manager has

also to decide which services to import according to a novel advertisement filtering mechanism. Such a mechanism takes into account both functional constraints and the current composition of the service registry. This mechanism is provided by the Registry Info class, and will be extensively described in Section 4.3.

4.2.2 Designing Master DSAs and building Federations

MDSA vs SDSA

The interface of a Master DSA is quite similar to that of a Slave DSA, even though several significant differences exist between the two agents. The main differences between the responsibilities of such agents are presented in Table 4.1: since the MDSA is a federation-level discovery agent, it does not depend on a specific interaction platform, and it is not concerned with the creation and destruction of service-specific proxies. Moreover, as the MDSA manages only Interoperable Service Description Records, it provides no translation or completion services.

Feature	SDSA	MDSA
Import/export	Within a nomadic domain	Within a domain-federation
Filtering	Yes	No
Translation	Yes	No
Proxy management	Yes	No
External advertisement	Federation-level	Global Environment level
Platform dependent	Yes	No

Table 4.1: Comparing the responsibilities of Master and Slave Domain Specific Agents

Designing intra-federation and inter-federation discovery protocols

As mentioned in Section 3.2.2, the Master DSA provides key discovery services at the Federation / Global Environment interface (*inter-federation discovery*), and within its own Federation as well (*intra-federation discovery*). Since in a real nomadic computing environment the number of agents involved in intra-federation and inter-federation discovery operations can be significantly large, it is necessary to use scalable communication protocols and paradigms to design and implement such discovery services. Such communication protocols and paradigms should also be suitable for achieving interoperable resource-aggregation.

In order to satisfy such requirements, a peer-to-peer decentralized approach is adopted for the communication between Master and Slave DSAs. Such an approach lays the groundwork for more robust, dynamic, and scalable search mechanisms [53]. Moreover, it is valuable for a variety of reasons, which are briefly discussed in the following. In a peer-to-peer model, different peers can be grouped into PeerGroups. Peers within the same PeerGroup can send both group-level and one-to-one messages to other peers. By adopting a peer-to-peer approach, each domain-federation can be assigned a certain PeerGroup, thus considering each Slave DSA within a certain federation as a peer belonging to the Federation's PeerGroup. Similarly, Master DSAs are designed as peers belonging both to the Global environment PeerGroup and to the Federation PeerGroup, as depicted in Figure 4.5.

Such a peer-to-peer model paves the way to the design of group-discovery and group-announcement protocols which present the following characteristics.

- *service registry*: each Slave DSA has its own service registry which contains the list of the Interoperable SDRs of its local services; similarly, each Master DSA

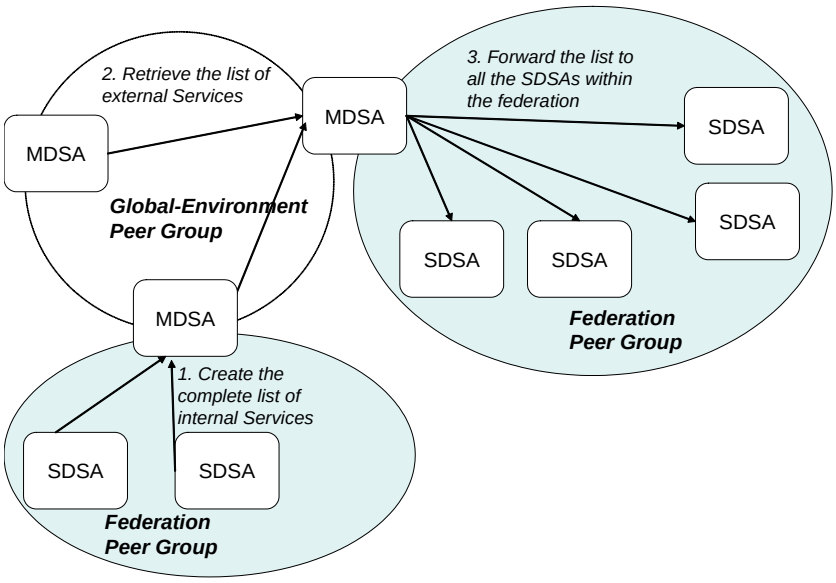

Figure 4.5: Peer-to-peer approach for Inter-federation discovery and for the subsequent advertisement of external services within a certain federation.

holds a list of all the services available in its own federation;

- *inter-federation service advertisement*: each Master DSA can retrieve the service-lists of all the Master DSAs registered with the Global Environment by sending a specific **reactive-discovery advertisement** to the Global Environment's Peer Group; upon the arrival of such an advertisement, each MDSA sends the list of its own federation to the requesting MDSA (see steps 1. and 2. in Figure 4.5);

- *proactive intra-federation service advertisement*: each Master DSA retrieves the service-lists of all its Slave Agents through periodic **service-list advertisements** sent to the federation's Peer Group; it subsequently uses them to build a complete list of all the services available within the Federation; such a complete list is then sent to all the Slave DSAs by means of a **service-list update**

advertisement within the Federation's PeerGroup. The same mechanism is also used to publish the service-descriptors received by means of an inter-federation service advertisement. Such a peer-to-peer mechanism for importing services into a federation is presented in Figure 4.5.

4.3 Proactive filtering of Service Advertisements

4.3.1 The filtering algorithm

Each Slave DSA holds a domain-level service list which comprises the Service Description Records of all the services currently published within the domain of its own. Import operations can significantly increase the dimension of such a service list, thereby increasing the dimension and the complexity of the resulting domain. Visibility rules provide a first step toward the management of domain complexity, even though they do not guarantee that domain resources be always available. Moreover, resource unavailability can force agents to discard new import operations; this can make entire service classes unavailable. Therefore, the Import Manager adopts a more complex filtering strategy, in order to guarantee that a minimum number of service records is assigned to each service class. As already mentioned such a mechanism is provided by the Registry Info component.

In order to build a precise model the proposed algorithm, it is necessary to give the following definitions. Let $D_M(D_1)$ be the maximum dimension of domain D_1, and let $C(D_1) = C_j(j = 1..n)$ be the set of service classes provided by D_1. Hereafter by $D_R(D_1)$ it is meant the residual dimension of D_1, whilst by $minspace(C_j)$ it is meant the minimum space required by class C_j. If $minspace(C_j)$ has been allocated

to each class, the number of C_j services can exceed $minspace(C_j)$. Thus, available classes can be categorized into Exceeding (Ex) and Not Exceeding ($NotEx$) classes, depending on the current number of services; thereby $N(D_1)$ is the overall number of service classes available in D_1, $N_{Ex}(D_1)$ and $N_{NotEx}(D_1)$ are the number of exceeding and not-exceeding classes respectively. $Dim_{Ex}(D_1)$ and $Dim_{NotEx}(D_1)$ represent the fraction of D_1 assigned to exceeding and not-exceeding classes respectively.

Given a service s_i, the filtering rules of the proposed proactive algorithm are defined as follows:

- if $C(s_i)$ is not exceeding, s_i is imported into LD_1;
- if $C(s_i)$ is exceeding, s_i is imported only if all the classes have their own *minspace* available, or equivalently:

$$D_R(D_1) \geq N_{NotEx}(D_1) \cdot minSpace - Dim_{NotEx}(D_1)$$

  being $N_{NotEx}(D_1) \cdot minSpace - Dim_{NotEx}(D_1)$ the space available for not-exceeding classes;
- if importing s_i empties the space available for not-exceeding classes, s_i is imported if and only if the following conditions hold: *a*) s_i is usable, and *b*) the service s_d to discard is either not usable or rarely used.

Filtering rules are depicted in Figure 4.6. As for the last rule, usability is the possibility to build service proxies on both client-side and server-side.

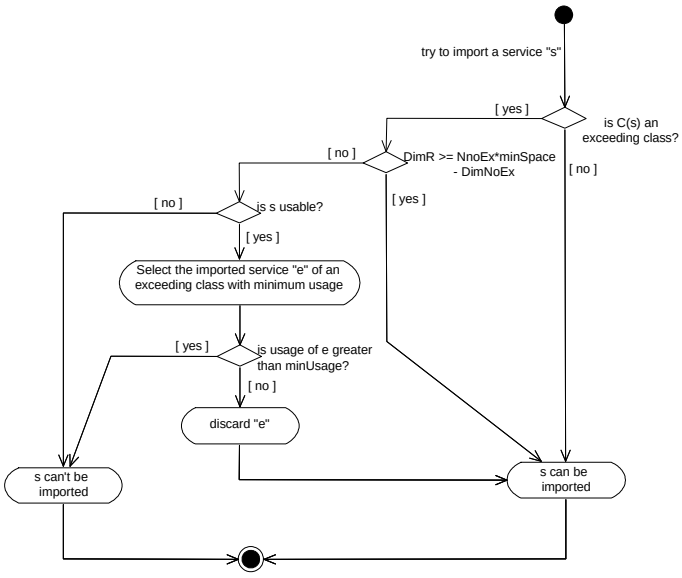

Figure 4.6: Filtering algorithm for proactive discovery

4.3.2 Evaluating the stability of the filtering mechanism

The filtering algorithm can limit the number of imported services N_e. Let N_{tot} be the total number of services to import, and N_{imp} be the number of effectively imported services. The stability of the filtering algorithm can be estimated through the *import ratio* t_i parameter, defined as follows: $t_i = \frac{N_{imp}}{N_{tot}}$. As an example, $t_i \approx 1$ means that i) remote domains behave irregularly, for they announce and discard services very frequently, and/or ii) services are exported before reaching a significant usage rate (i.e., the import/export mechanism is too fast with respect to client requests). Hence, an unstable mechanism is characterised by $t_i \approx 1$, i.e., a not-optimal service usage is being performed. Conversely, $t_i \approx k, 0 < k < 1$ means that services are effectively used, and service discarding is quite a rare event, thus achieving import mechanism stability.

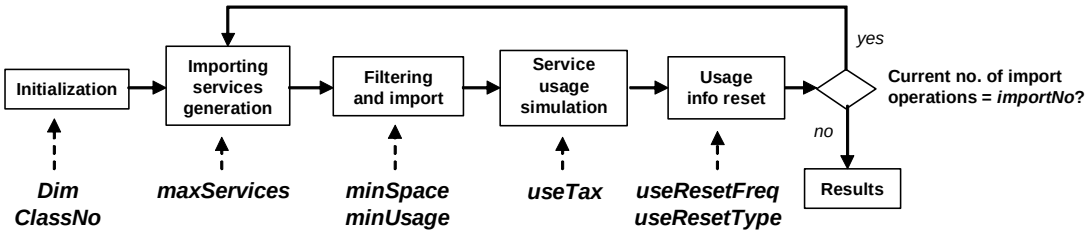

Figure 4.7: Simulation model of the filtering mechanism

The stability of the discovery mechanism has been evaluated by means of a MATLAB simulation model. This model relies on the following assumptions: $a1$) each service is represented by filtering parameters, i.e., service class and service usage; $a2$) all the services to import are visible; $a3$) all the services to import are usable; and $a4$) remote domains behaviour is regular. Figure 4.7 depicts the simulation model used for evaluating the filtering mechanism. The model can be tuned by setting two main types of parameters, namely *internal parameters*, which directly characterise the filtering mechanism, and *external parameters*, which significantly affect the mechanism, even though they do not belong to it.

External parameters include the registry dimension, Dim, the overall number of classes, $ClassNo$, the maximum allocation vector, $MaxServices(i)$ ($MaxServices(i) = n$ means that each import operation should import at most n services of class i), and the service-usage vector, $UsageRate(i)$ ($UsageRate(i) = t_{av}$ means that t_{av} is the average usage-time of class-i services). **Internal parameters** comprise the minimum space assigned to each service class, $MinSpace$, the minimum usage threshold, $MinUsage$, the frequency of updates to the service-usage information, $UsageResetFreq$, and the type of update to perform, $UsageResetType$,i.e., reset usage or divide current usage by the specified value.

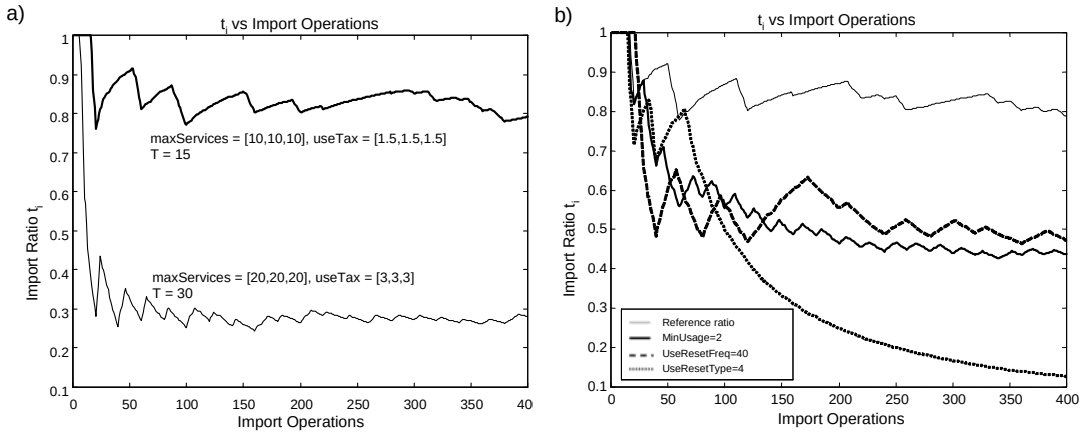

Figure 4.8: Effects of filtering parameters on the import ratio

Given the inter-filtering time T and the quantity of announced services Q, the average volume of services is defined as $V = T \cdot Q$. However, V can also be calculated as follows: $V = \sum_{i=1}^{i=ClassNo} \frac{MaxServices(i)}{2}$, since $MaxServices(i)$ elements are uniformly distributed. Hence, V can be tuned by setting $MaxServices$'s components.

The stability of the import mechanism has been evaluated with respect to the inter-filtering time T. For the sake of simplicity, the agent is assumed to import one service each second (i.e., Q=1). Figure 4.8a) shows that, when $T = 30$, or equivalently when $V = TQ = 30$, the steady-state import ratio is close to 30%; if T is halved, the import mechanism gets unstable; this instability is represented by high values of the import-ratio, which reaches $80 - 90\%$. Hence, Figure 4.8a) shows that increasing T can enhance import stability. If the mechanism has to operate in an environment characterised by low values of T, a different strategy is needed. Figure 4.8b) shows how to tune internal parameters so as to enhance system stability upon low inter-filtering times; the reference import-ratio represents an unstable mechanism, characterised by $MinUsage = 3, UseResetFreq = 20, UseResetType = 0$. By forcing $MinUsage = 2$,

the mechanism behaves more selectively, since services that had been used at least 2 times (upon the last registry reset operation) cannot be discarded; therefore, the import ratio value is lower than the reference one, i.e., the mechanism is more stable. Stability can also be enhanced by varying the characteristics of the registry-reset operations. Increasing $UseResetFreq$ up to 40 can specifically lead to a more stable mechanism. Initial variations of the import ratio can be smoothed by decreasing the $MinUsage$ parameter. The last curve represents the effects of changes in the $UseResetType$ parameter: by dividing usage information by 4 at each reset operation ($UseResetType = 4$), the resulting curve is more regular and asymptotically reaches a zero value. Hence, in this case the higher the number of requested operations, the more selective the mechanism is.

4.3.3 Using the proactive mechanism for importing external services

In order to show how to use the proposed algorithm as a strategy for SDSA to import external services, the rest of this section outlines the overall architecture of the Registry Info component. The components involved in proactive filtering are presented in Figure 4.9.

The **Local Registry Info** is the main class within this component. It holds the list of all the service classes that a certain domain can provide; such a list is represented as a list of ServiceClassInfo objects. As regards the management of the filtering mechanism, the Local Registry Info implements two different interfaces, namely IRegistryInfoControl and IRegistryInfo. While the former is used only to set the internal parameters of the ISDR registry, the latter is used to retrieve the

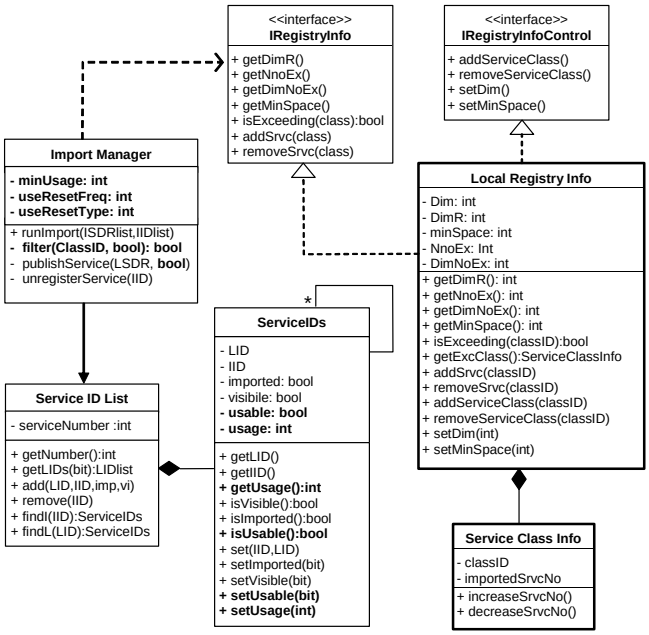

Figure 4.9: UML Class Diagram of the Registry Info Component

information required by the filtering algorithm, and to update the registry when publishing or unregistering services.

4.4 Implementation Issues

Figure 4.10 depicts the overall architecture of the implemented prototype. Two different Slave DSAs have been implemented, in order to use the prototype to integrate Jini and Bluetooth domains. The component diagram in Figure 4.10 shows that the prototype relies on several COTS component. Each SDSA interacts with its child domain by means of the functionality provided by a technology-specific API. As for Bluetooth domain, the Official Linux Bluetooth protocol stack (BLUEZ) [6] has been adopted. This library provides quite a comprehensive implementation of a Bluetooth

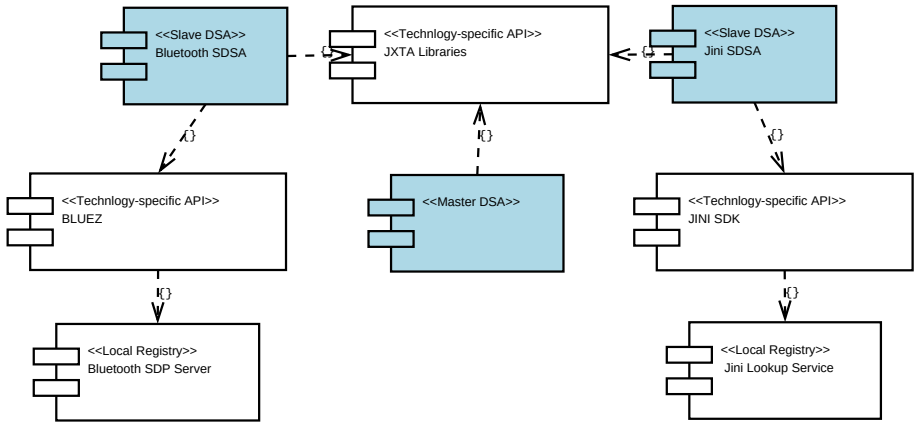

Figure 4.10: High-level components of the infrastructure prototype. Grey boxes represent the implemented components.

Stack to Linux Applications written in the C and C++ languages. As for the Jini domain, the Jini SDSA uses the Jini Technology Starter Kit 2.0 libraries.

As for peer-to-peer communication, the inter- and intra-federation discovery protocols require a scalable, dynamic, and flexible mechanism for peers to communicate within the global environment. JXTA is an open source technology which has been designed to build dynamic large-scale peer-to-peer computing systems. Since its suitability to address such issues has been demonstrated by a comprehensive evaluation [55], the implemented intra-agent discovery protocols use it as a peer-to-peer messaging and collaboration layer.

It is worth noting that the implemented MDSA is completely independent of the platforms to integrate. Conversely, the two SDSAs are tightly related to their own target platform (i.e., Jini or Bluetooth). For the sake of simplicity, the rest of this Section describes the implementation of a Slave DSA component with respect to the Bluetooth domain. However, the analogies and differences between the two agents are pointed out explicitly while describing the Bluetooth Slave DSA.

4.4.1 Implementing a Slave DSA for Bluetooth domains

Translating Bluetooth Service Records into Interoperable SDRs

Figure 4.11 compares a typical Bluetooth Service Descriptor with an Interoperable Service Description Record.

More specifically, it shows that the semantic matching condition holds only for a small sub-set of Bluetooth description fields; moreover, it also shows that the obligatory nature of certain fields depends on the particular domain. For instance, the Service ID and Service Name fields are optional in Bluetooth Descriptors, whereas they are mandatory in Interoperable Service Descriptions Records. The Mandatory Attribute class provides all the methods to fill the obligatory attributes in the involved service descriptors.

The implementation of the Description Translator, depicted in Figure 4.12, reflects the relationships between the service descriptors in Figure 4.11.

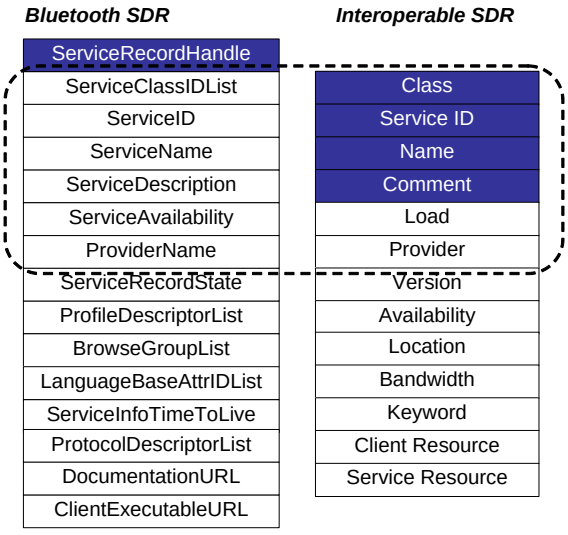

Figure 4.11: Mapping a Bluetooth Service Descriptor onto an Interoperable Service Description Record. Dark boxes represent the mandatory attributes.

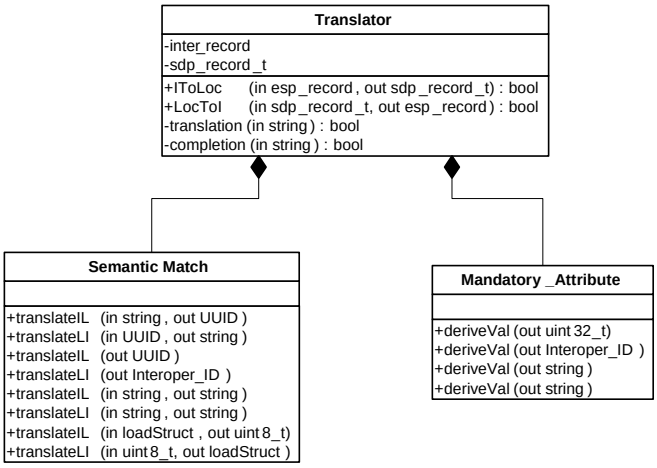

Figure 4.12: UML Class diagram of the Bluetooth Description Translator component.

Indeed, the Semantic Matching class provides all the methods required to translate the fields from the native Bluetooth representation to the Interoperable SDR format.

For instance, the `translate_IL(in String, out UUID)` method is used to translate the the Class field of the ISDR into the ServiceClassIDList field of a Bluetooth SDR. The reverse translation is provided by the `translate_LI(in UUID, out String)` method.

While the translation of the Class field is quite straightforward, the translation of different fields may require several operations to be done. For instance, Bluetooh ServiceAvailability (av) measures service load in a range between 0 and 255 (e.g. sA=25 means that about 90% of service capacity has been allocated to active clients), while the ISDR Load parameter represents the same concept by means of a structured information (called `LoadStruct`). A `LoadStruct` is composed of two sub-fields, namely `client_no` (i.e. the number of active clients) and `maxClients` (i.e., the maximum number of allowed clients for a certain service). The

`translate_IL(in load:loadStruct, out av: uint8_t)` method provides a valid Bluetooth Availability field by processing the Load field of the Interoperable SDR. More specifically, this method calculates availability as the minimun integer number not greater than the following value: $\left(1 - \frac{load.client\_no}{load.maxClients} \cdot 255\right)$.

As for the Jini Slave DSA, the translation is more flexible since each Jini service is described by means of an associated Entry Object. The structure of this Entry can be personalized and extended seamlessly. Hence, it is possible to translate every Interoperable SDR into a jini Entry Object. Unfortunately, the reverse translation is not as flexible as the Jini to ISDR one, since it could not be possible to identify the service class of a Jini service. If it is possible to modify a service Registration Entry by adding an `ISDR_ServClass` attribute to it, the agents will be able to recognize it so as to classify the service correctly. Conversely, most of legacy Jini services will be imported as Generic Services.

Interacting with local registries

The overall architecture of the Bluetooth SDSA comprises both Java and C++ components, as Figure 4.13 shows. The bridge between the two different types of component has been established by means of the Java Native Interface (JNI) mechanism. The SDSA_Peer is a Java-based component which on the one hand is in charge of interacting with other agents by means of the implemented JXTA-based intra-agent protocols, while on the other hand it exploits lower level C++ components (such as the Import Manager and the Export Manager) in order to deal with all the domain-management issues that have been described in Section 4.2.

The UML sequence diagram of an import operation is presented in Figure 4.14.

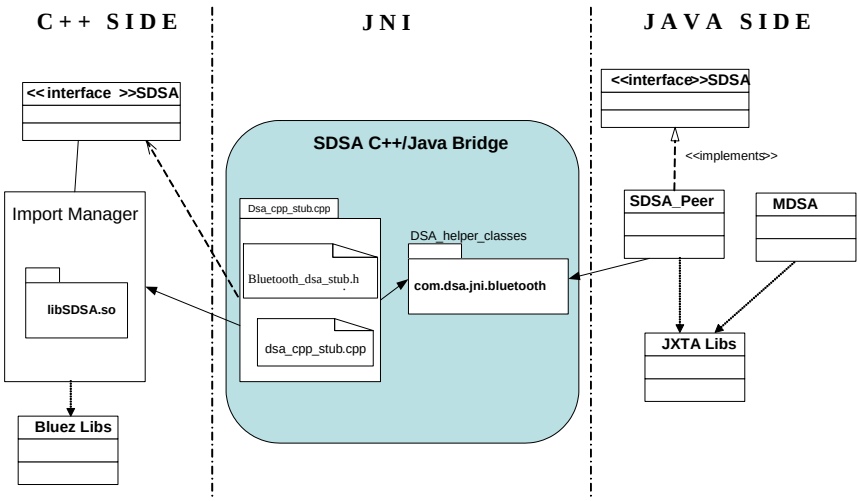

Figure 4.13: Integrating Java and C++ components of the Bluetooth Slave DSA

Such a scenario shows that the SDSA_Peer wraps the Import Manager component in order to trigger its native methods upon the arrival of JXTA advertisements from its own Master DSA.

As regards the interaction of the Jini SDSA with a Jini Lookup Service, the overall architecture of the implemented agents is much more simple than that of the Bluetooth SDSA. Indeed, since the Jini Technology Starter Kit consists of a set of Java Packages, no JNI bridging mechanism is required, thus eliminating the need to implement the additional SDSA_Peer component. The Import Manager component can join the JXTA PeerGroup directly in order to receive and send messages to its Peers.

4.4.2 Using the JXTA technology to build domain-federations

The adoption of the JXTA technology as an effective communication layer for peer-to-peer interaction between agents requires to map agents and service-provisioning

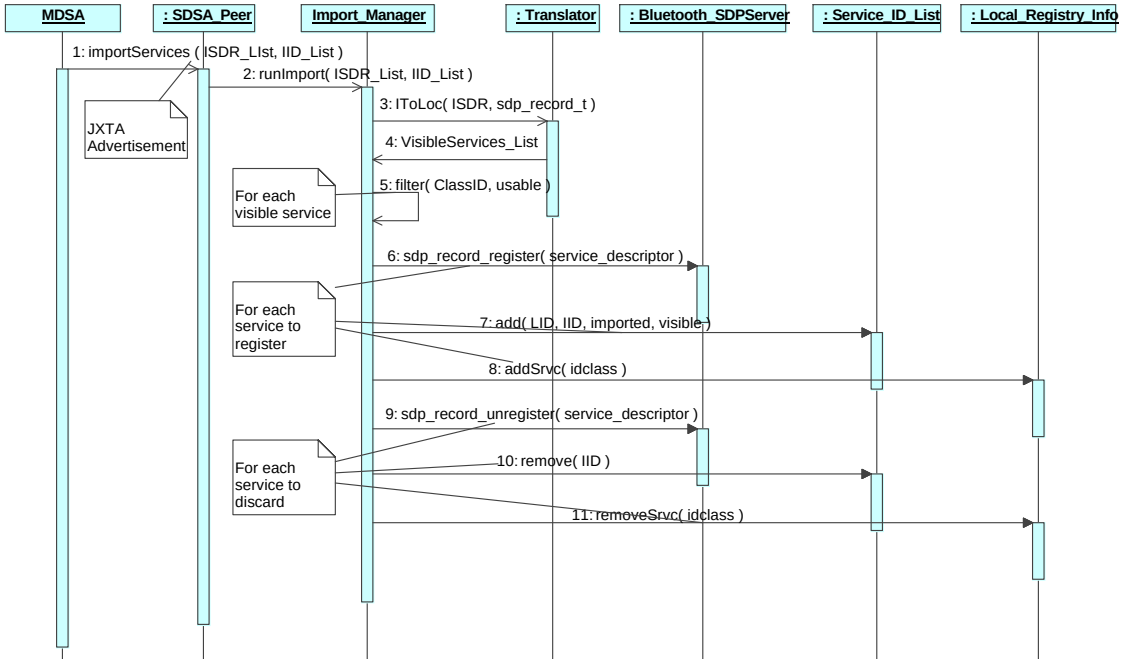

Figure 4.14: Importing services into a Bluetooth domain.

domains to JXTA entities and concepts. The interested reader may refer to [59] for further details about such concepts. It is worth noting that JXTA is an hybrid peer-to-peer technology, since its messaging and routing facilities are based on the presence of different classes of Peers. A basic peer can always receive and send messages to either another peer or an entire PeerGroup.

A special class of peers called **rendezvous peers**, function as Super-node for a JXTA system. That is to say, it allows other peers to discover advertisements from the rest of the network. The rendezvous can also delegate queries to other peers, which must also be a rendezvous. Edge peers publish indexes of advertisements across Rendezvous network using Distributed Hash Tables (DHT). DHTs are maintained by Rendezvous peers and are based on pluggable Hash functions. A **relay** peer is a special rendezvous that can forward both requests and messages.

In the proposed infrastructure, each Master DSA joins two different PeerGroups in compliance with the conceptual schema depicted in Figure 4.5. The first PeerGroup represents the global-environment and comprises all the available Master DSAs. The second PeerGroup, representing the federation of the specific MDSA, comprises the Slave DSAs assigned to each nomadic domain. The Slave DSAs discover their Master by means of PeerGroup advertisements. Each Master DSA functions as rendezvous for all the Slave DSAs comprised in its own federation.

4.4.3 Master DSAs and inter-agent communication

In order to connect the Master DSA to JXTA PeerGroups, several components have been introduced into its architecture, as Figure 4.15 shows. The interaction between agents during intra-federation and inter-federation discovery protocols is built upon bidirectional JXTA pipes. A set of basic messages has been defined in order to implement such discovery protocols. Such messages are encapsulated into an helper Java interface, called DSAMessages, which must be implemented by the involved parties. A custom PipeAdvertisement, called DSAPipeAdvertisement, has also been defined. As for the management of inter-federation and intra-federation discovery protocols, the MDSA delegates most of the work to three components, namely the **ConnectionThread**, the **DiscoProactiveThread**, and the **ReceiveAndSendThread**. Such components allow the MDSA to receive and send advertisements to its peers in parallel. They implement the JXTA_PipeMsgListener interface, which allows agents to interact with each other asynchronously by means JXTA Bidirectional-Pipes.

The **ConnectionThread** component is in charge of creating JXTA advertisements for sending specific requests from a Master DSA to either its Slave DSAs or

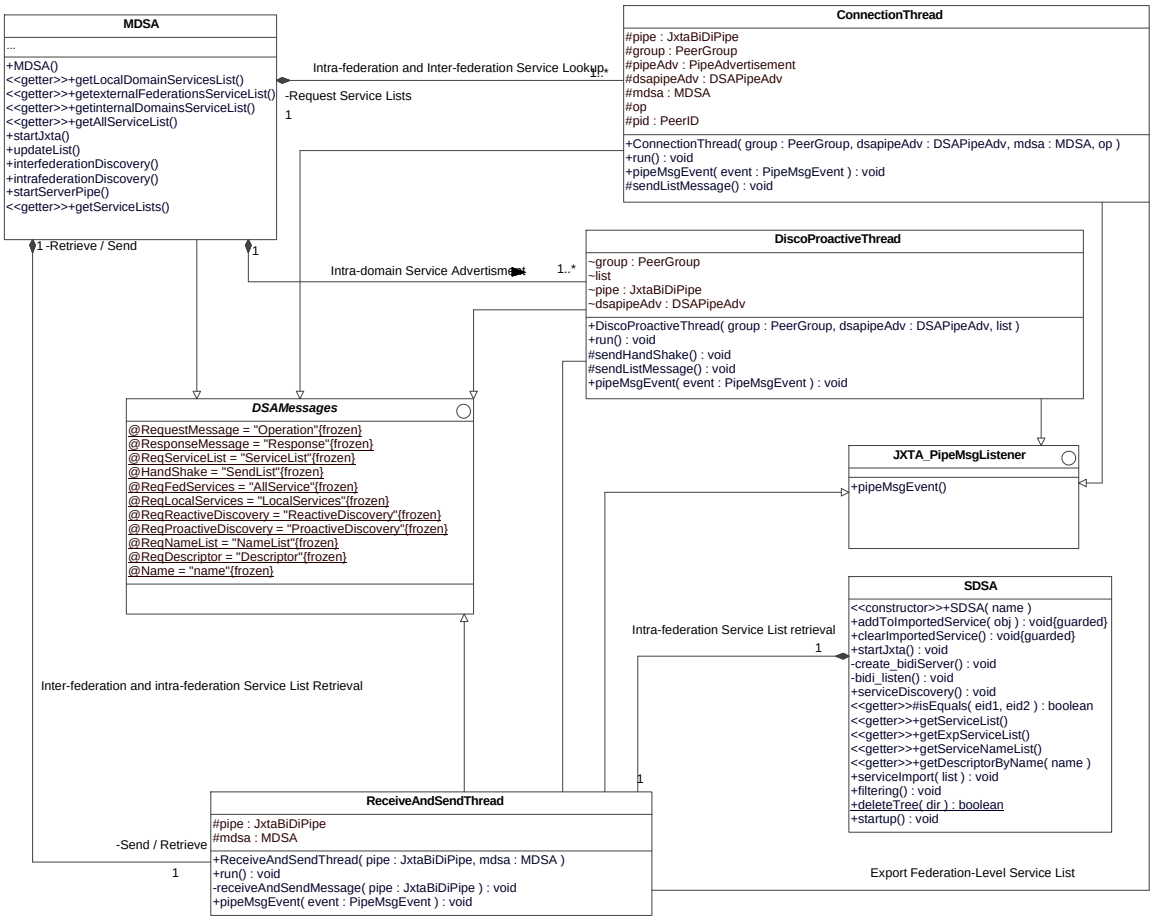

Figure 4.15: UML Class Diagram of the implemented Master DSA

other Masters within the global environment. Hence it is involved in the early steps of both intra-federation and inter-federation service discovery.

The **ReceiveAndSendThread** is responsible for answering to incoming requests. Both the SDSA and the MDSA hold a ReceiveAndSendThread object which function as a broker for messages on JXTA Pipes. Such an object is in charge of parsing and decoding the incoming requests, and of triggering the required operations accordingly. For instance, as regards to intra-federation discovery, the MDSA keeps track of all the Slave DSAs that joined its Federation PeerGroup. It uses such a peer-list to contact its

Slave DSAs periodically in order to retrieve the service list of each domain comprised in its federation. To this aim, it creates one ConnectionThread object for each active SDSA, in order to send multiple service-list requests in parallel. The contacted SDSAs will exploit their own ReceiveAndSendThread object to answer to such requests. The MDSA's ReceiveAndSendThread object will gather all the incoming answers in order to update the MDSA's Federation-level service list (*Intra-federation Service Lookup*).

The **DiscoProactiveThread** component is in charge of forwarding the Federation-level service list to each SDSA in a certain federation. In order to proactively update the service-list of each Slave DSA comprised in the Federation (*Intra-federation Service Advertisement*), the MDSA assigns one DiscoProactiveThread object to each discovered SDSA; each object will periodically send the federation-level list to its own SDSA's ReceiveAndSendThread object.

4.5 Putting all the pieces together: the interoperable printing service

In order to evaluate the effectiveness of the proposed interworking approach, the implemented prototype has been used to leverage the interoperability of a real bluetooth service. More specifically, a print service has been used as an example.

The architecture of the experimental testbed is depicted in Figure 4.16. The considered global environment consists of two different federations, each assigned to a specific laboratory within my research group, namely the Computer Science Department Lab and the CINI-ITEM Lab.

The laboratories are located in two different buildings and belong to different

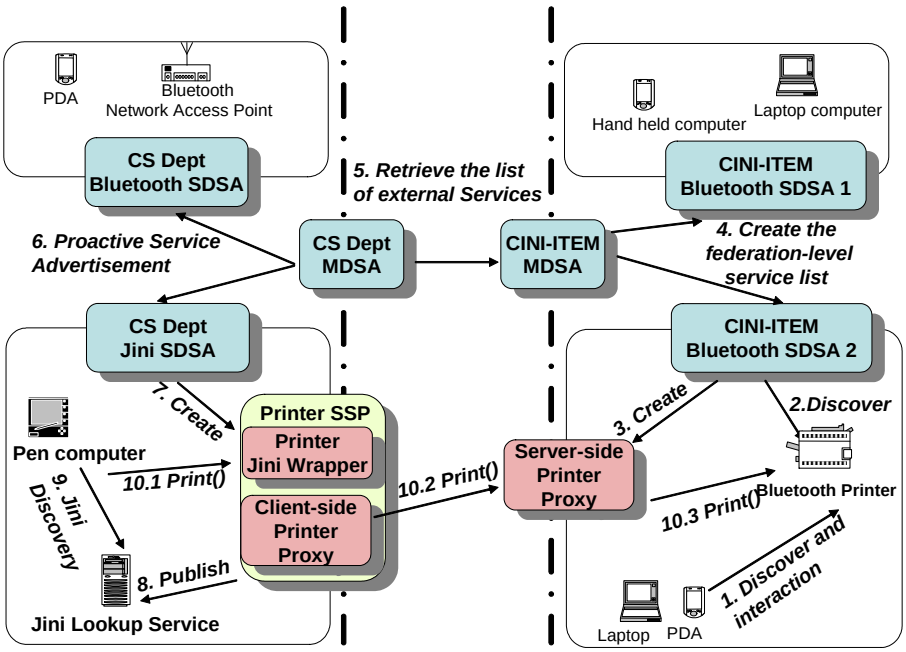

Figure 4.16: Overall architecture of the implemented case-study environment

organizations. The CSDept Federation comprises a bluetooth piconet and a Jini federation. The CINI-ITEM federation consists of two bluetooth piconets. The CINI-ITEM Federation 2 comprises a Bluetooth printer. Due to the lack of interoperability between bluetooth and Jini, the Jini-enabled laptops cannot discover (and use) the bluetooth printer.

By using the proposed interworking infrastructure, the Bluetooth printer has been exported to external domains. Such an export process consists of several steps, which are depicted in Figure 4.16. Such steps are briefly described in the following.

1.Bluetooth-based terminals can discover and use the Bluetooth Printer by means of direct bluetooth-based interaction.

2. - 3. The bluetooth Slave DSA discovers the bluetooth printer and creates the related server-side Service-Specific Proxy.

4. CINI-ITEM's Master DSA builds the federation-level service list.

5. - 6. The CS Department's Master DSA retrieves the list of external services, and sends it to its own Slave DSAs.

7. - 8. The Slave DSA creates the client-side service proxy for the Printing service. Such a SSP consists of two components, namely a client-side ssp and a jini wrapper. The former is in charge of interacting with the server-side SSP, whereas the latter is responsible for service registration with the Jini Lookup Service, in order to be discovered by Jini clients as a standard Jini Service.

9. A mobile user looks the available services up for a printing service, and can find it thanks to the presence of the Printer SSP.

10. The printing requests sent by the pen computer (10.1) are translated and forwarded to the server-side SSP (10.2), which subsequently sends the data to the printer by means of a bluetooth-based interaction (10.3).

It is worth noting that when a user looks services up from the Jini-enabled pen computer, the imported printing service is presented as an active Jini Service. Hence the user can discover and use it as an off-the-shelf Jini Service.

To give a visual understanding of the interworking capability of the infrastructure on the service-discovery layer, an environment-browsing tool has been developed. Such a tool allows to browse federations in order to give users an overall view of the current composition of the global environment. This tool allows to see which federations are active, and allows to browse the available agents within each federation. It is also possible to query each agent in order to retrieve the list of services available within its own service-provisioning domain. More specifically, it is possible to retrieve the federation-level service list by asking a specific MDSA to send its own

service list. Similarly, it is possible to retrieve a domain-level service list by asking a specific SDSA to send its own service list.

Chapter 5

Addressing Unpredictability and Context-awareness: two case-studies

The previous chapter showed how to address the scalability and interoperability issues which stem from the strong heterogeneity of nomadic environments. This chapter focuses on unpredictability and context-awareness issues of two particular types of application. More specifically, this chapter describes the implementation of two service elements. The first supports nomadic multimedia streaming applications and relies on a novel technique to address unpredictable performance of nomadic multimedia streaming. The second supports location-aware applications for Java-based mobile devices and relies on an innovative approach for indoor-location-sensing.

5.1 Addressing the unpredictable behavior of Nomadic Multimedia Services

5.1.1 Dependable Multimedia and Unpredictability

Mobile multimedia applications are becoming of increasing interest to applications with stringent dependability requirements, such as internet robotics, flight control, telemicroscopy, telemedicine, and military applications [34, 60]. The unpredictable behaviour of the exploited multimedia stream can significantly affect the dependability of the resulting application. Addressing the unpredictability of real-time streaming service on mobile devices is the most crucial issue that these nomadic computing applications must face so far. Hence, it is crucial that a generic service element be provided so as to provide such applications with a failure-detection mechanism. Although a great deal of research has been conducted on dependability of multimedia applications, to the best of our knowledge, an off-the-shelf service element for achieving fault tolerance in the considered scenario has not been proposed yet.

This section presents the design and the implementation of a generic service element to support fault-tolerant distributed multimedia applications that rely on the standard Real-time Transport Protocol (RTP) [54]. Realizing a fault tolerance strategy for such applications is quite a complex task, since a variety of unpredictable factors must be taken into account, such as the error detection process (e.g. network performance, software, and hardware errors), the recovery strategies (based on detection), the multimedia encoding format, and the perceptual quality. The proposed service element faces such an unpredictability by means of a novel strategy to model meaningful system failures in terms of multimedia stream performance parameters (i.e., delay, delay variation, and information loss).

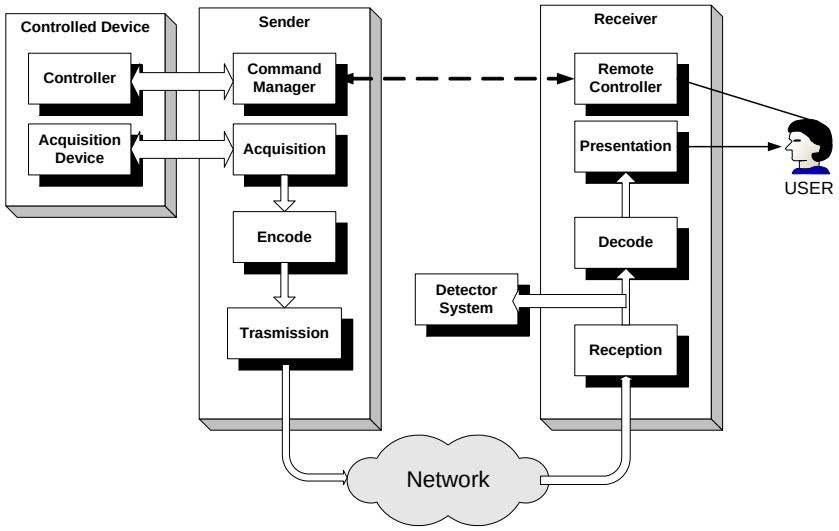

Figure 5.1: Conceptual model of nomadic distributed multimedia systems

The generic service element refers to distributed multimedia systems for video-based remote control applications. The conceptual model of such a system is depicted in Figure 5.1. It consists of the following components: *i*) the device to control; *ii*) the sender of the multimedia streams that are used to control the device; *iii*) the receiver; and *iv*) a wireless communication channel (e.g. Bluetooth and/or Wi-Fi). The sender might physically reside either on the controlled device or on the control host. The user controls the device by forcing movements through the remote controller, based on multimedia data captured by the acquisition device.

The rest of this section proposes a novel service element which transparently enables failure detection and monitoring of real-time multimedia streams. Hence, it can be used to build fault-tolerant strategies for off-the-shelf multimedia services, or more generally for real-time services that use the RTP protocol as a transport layer.

5.1.2 Preliminaries

The proposed service element relies on a threshold based mechanism to discriminate faults. More specifically, it is based on the alpha-count family of mechanisms, since it provides a simple but effective strategy to identify different classes of fault. The basic alpha-count mechanism is a count-and-threshold scheme originally devised "to consolidate identification of faults, distinguished as transient or permanent/intermittent" [7]. The simplicity of the alpha-count mechanism allows to implement it as a small, low-overhead and low-cost module (suitable even for embedded real-time systems). Furthermore its simplicity makes it also easy to explore its behavior and its effects on the system by analytical means.

The count and threshold mechanism relies on an error signaling module that collects error signals from any error detection device in the system. Error detection results are periodically delivered to the alpha-count mechanism as binary signals. The judgment on a component's behavior, given by the error signaling mechanism, is correct with a probability, or coverage, c. The alpha-count processes information about erroneous behavior of each system component by giving a smaller weight to error signals as they get older. Each not-yet-removed component i is assigned a score variable α_i in order to record information about the errors experienced by that component. α_i is initially set to 0, and accounts for the L-th judgement as follows:

- $\alpha_i(L) = \alpha_i(L-1) + 1$ *if the i-th component is marked as faulty at the i-th execution steps*
- $\alpha_i(L) = K * \alpha_i(L-1)$ *if the i-th component is marked as correct at the i-th execution step*

where $0 < K < 1$.

If $\alpha_i(L)$ becomes greater than or equal to a given threshold α_T, the i-th component is marked as failed and a signal is raised to trigger further actions (error processing or fault treatment). The effectiveness of the mechanism depends on K and α_T. The optimal tuning of these parameters depends on the expected frequency of errors, and on the probability c of correct judgments of the error signaling mechanism. The analysis in [7] showed the trade-off between delay and accuracy of the diagnosis; it also showed how to tune these parameters for optimizing the behavior of the detection mechanism.

5.1.3 Failure modes of multimedia services

A multimedia service s may be considered as the aggregation of multiple multimedia streams (e.g., an audio track and a video track). Each multimedia stream i is characterized by the following three temporal distributions:

1. *delay distribution*, contains a sequence of values $td_i(n)$, where $td_i(n)$ represents the delay of the n-*th* packet; a maximum delay threshold D_{max} is used to distinguish delayed packet from the dropped ones;

2. *delay variation distribution*, contains a sequence of values $tdv_i(n) = td_i(n) - td_i(n-1)$, where $tdv_i(n)$ represents the delay variation of the n-*th* packet;

3. *information loss distribution*, contains a sequence of values $til_i(n)$, representing the percentage of packets that have been delayed more than D_{max} in an observation period T. Such values are calculated as follows:

$$til_i(n) = \frac{\sum_{j=n-T}^{n} tpl_i(j)}{n}$$

being:

$$tpl_i(j) = \begin{cases} 1 & \text{if } td_i(n) > D_{max} \\ 0 & \text{otherwise.} \end{cases}$$

It is worth noting that delay distribution can influence other parameters; delay may vary according to either information processing or transmission events.

The proposed service-element allows to detect several classes of failures of the multimedia stream (stream-failures). In the rest of this sub-section several failure modes are defined according to the definitions given in [45]. Value errors are not taken into account. Indeed, such errors occur when data gets corrupted while traversing from the server to the client; multimedia applications can tolerate such errors under many circumstances [57]. Let STD_i and $STDV_i$ be the subsets of all delay and delay variation distributions, respectively, that do not compromise the quality of the multimedia content. These subsets can be defined by either empirical evaluation or video-quality measurement. Four classes of stream-failures can be defined as follows:

- `delay failure`, f_d, occurs when the following condition holds: $(td_i \notin STD_i)$. It is worth noting that a `crash failure` is a particular case of the delay failure. Indeed, a crash failure occurs when $(td_i \in STCF_i \subset \neg STD_i)$, being the $STCF_i = \{\exists k : \forall j \geq k, td_i(j) \geq D_{max}\}$;

- `delay variation failure`, f_{dv}, occurs when the following condition holds: $(tdv_i \notin STDV_i)$.

Media type	Application	Bandwidth	Quality Parameters		
			Delay	Delay variation	Information loss
Audio	**Conversational**	4-64 kb/s	<150msec <400msec	<1msec	<3% PLR
Audio	**Messaging**	4-32 kb/s	<1 s <2 s	<1msec	<3% PLR
Audio	**High quality audio Streaming**	16-128 kb/s	<10 s	<1msec	<1% PLR
Video	**Videophone**	16-384 kb/s	<150msec <400msec	-	<1% PLR
Video	**One-way**	16-384 kb/s	<10 s	-	<1% PLR

Figure 5.2: Multimedia Application requirements

- `information loss failure`, f_{il}, occurs when $(til_i \geq qil_i)$, being qil_i the maximum percentage of lost packets that is acceptable to the specific application;

- `transparent failure`, f_{trp}. All the mentioned parameters td_i, tdv_i, til_i can lightly worse at the same time. Such multiple degradations can affect the quality of the multimedia content thus precluding the user to control the device anymore, even though application-requirements are fulfilled by each single parameter. To model such a failure-event a new class of failure, called transparent failure, must be defined. Being $np_i = (td_i, tdv_i, til_i)$, i.e., np_i is a point of the three dimensional space $CV = STDT_i \times STDVT_i \times \{q \in R : 0 \leq q \leq q_{il}\}$, where $STDT_i \subset STD_i$ and $STDVT_i \subset STDV_i$. A transparent failure occurs when $np_i \in CV$.

The correctness of a multimedia service can be now defined as follows: *a multimedia service is defined to be correct if none of the above mentioned failures occur.*

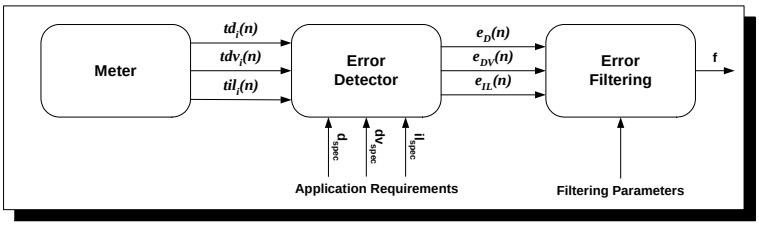

Figure 5.3: Conceptual model of the failure-detection service-element

5.1.4 The stream-failure-detection service-element

In order to design the failure-detection service-element, it is crucial that certain key issues be addressed. In particular, being STD_i and $STDV_i$ not computable in practice, we evaluate two alternatives: *i*) to find an heuristic strategy which attempts to individualize STD_i and $STDV_i$; *ii*) to choose an efficient and configurable schema which is able to detect failures with a high coverage. Due to the variable characteristics of multimedia contents and environmental conditions, the first alternative is not feasible. Hence, the rest of this section investigates the design and the implementation of a schema based on error filtering functions. Errors occur when the current values of $td_i(n)$, $tdv_i(n)$, and $til_i(n)$ do not satisfy application requirements. Indeed, as suggested by the ITU-T and represented in Figure 5.2, multimedia applications requirements can be modeled by three values d_{spec}, dv_{spec}, and il_{spec}, which represent the maximum values for delay, delay variation, and information loss, that are acceptable to each group of applications.

The main idea behind the proposed service-element is to monitor and process such parameters at run-time, using a combination of alpha-count functions. These functions can be configured in order to find a trade-off between the accuracy and the detection time, as described later in this section. Figure 5.3 depicts the conceptual model of the proposed detection system. It consists of three subsystems:

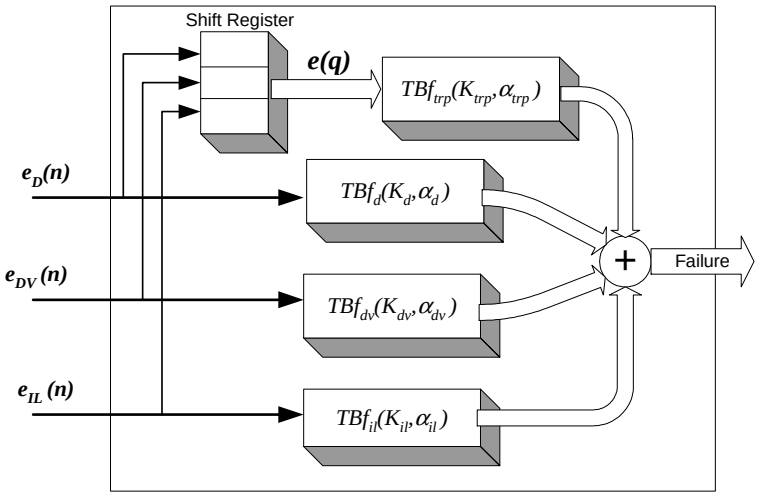

Figure 5.4: Error filtering subsystem

- `Meter`, which is in charge of measuring the quality parameters for each packet;
- `Error Detector`, which is responsible for the detection of three different errors: *i*) *error on delay*, defined by a boolean value $e_D(n) = (td_i(n) > d_{spec})$; *ii*) *error on delay variation*, $e_{DV}(n) = (tdv_i(n) > dv_{spec})$; and *iii*) *error on information loss*, $e_{IL}(n) = (til_i(n) > il_{spec})$;
- `Error filtering`, which is in charge of filtering the incoming error signals in order to detect failures.

The error filtering subsystem exploits a combination of threshold-based mechanisms based on alpha-count [7], as Figure 5.4 shows. In order to design such a subsystem, it is necessary that the following issues be addressed: *i*) detecting failures f_d, f_{dv}, and f_{il}, i.e. failures which stem from the degradation of one parameter; and *ii*) detecting failure f_{trp}, i.e. the defined *transparent failure*. In order to address the first issue, the service-element uses three separate alpha-count functions, TBf_d, TBf_{dv}, and TBf_{il}, each providing detection of failures on a single parameter. By tuning

these functions it is possible to configure the accuracy and the detection time of each single detector. An additional alpha-count module TBf_{trp} that receives all the error signals is used to detect transparent failures. Error signals, arriving concurrently from the three error signaling modules, are serialized by means of a parallel-in, serial-out register. It is worth noting that the frequency of this additional alpha-count is three times greater than that of the input signals.

As defined in [8], two different metrics can be used to evaluate a failure detector's behavior, namely *i*) the detection time (how fast the failure detector detects failures), and *ii*) the accuracy (how well it avoids mistakes). Preliminary simulation results – obtained by modeling the error filtering subsystem in the MATLAB environment – are presented in the following. The model receives real-world traces of multimedia streams as inputs. Data are stored in files with different multimedia formats (see Figure 5.8 in the next section). Three filter modules have been implemented in order to generate delay, delay variation, and information loss distributions artificially. Two are the outputs of the simulation models: the failure signal, which is true if a failure has been detected, and the filtered multimedia data.

Such a model allows the accuracy of the fault detector to be heuristically evaluated. To this purpose, we developed a multimedia player module that receives the failure signal, as produced by the model, and the filtered multimedia data. This module automatically presents multimedia data for a time interval where the failure has been detected. This allows us to check if a false detection occurs and to perform a better tuning of the filtering parameters.

The configuration of the alpha-count TBf_{trp} is not straightforward, for it depends on the characteristic of multimedia format. Indeed, multimedia applications can be

characterized by a Constant Bit Rate (CBR) or Variable Bit Rate (VBR) flows. One drawback of using VBR sources is the increased possibility of packet loss. This is due to high peak-to-mean ratios and significantly high autocorrelations, which characterize these sources. Such sources can result in very high values for delay variation if network resources are uncorrectly allocated. Using VBR flows, a transparent failure occurs when $tdv_i(n)$ and $til_i(n)$ degrade at same time, i.e., for a given packet n, $e_{dv}(n) = e_{il}(n) = 1$. CBR flows allow to simplify bandwidth allocation, and also to render the video source more suitable for traffic policing. However, CBR encoding has the drawback that video quality (distortion) varies significantly in order to ensure a constant bit-rate. Using CBR, a transparent failure occurs more likely when all the three parameters degrade at same time. In order to validate this assumption, the model has been tested also with simulated VBR (CBR) traffic by means of the Ns-2 network simulator; the outcomes of such tests were compliant with the mentioned assumptions.

Figure 5.5 shows the shape of the alpha-count functions with respect to a VBR video streaming. Four transparent failure have been raised by the TBf_{trp}. K and α thresholds were set to 0.7 and 2.5 respectively. It is worth noting that the video quality crucially worses upon the raised failures.

The accuracy and the speed of the count-and-threshold mechanism can be customized by varying filtering parameters. The configuration of the alpha-counts is performed according to the suggestions given in [7], which provided an exhaustive analysis of the effects of these parameters on detector's behavior. Simulation results indicated that, for the considered multimedia applications, lower values of k are needed. In fact, K represents the time window where memory of previous errors

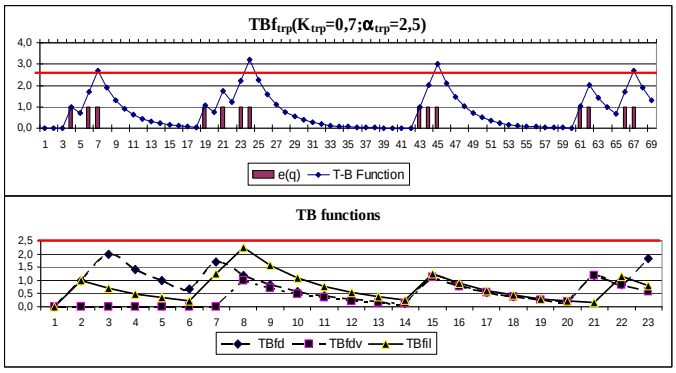

Figure 5.5: Transparent failure occurrences

is retained; choosing k between 0.7 and 0.8 results in a time window which allows to keep track of previous errors without slowing too much the decrease of the alpha function.

5.1.5 The case study application

Overall architecture

The proposed service-element has been implemented and tested with a case-study application, consisting of real-time control of a Lego Mindstorm [62] robot. Such an application allows to control the so-called "RoverBot", described in the Lego MindStorms "Constructopedia". The RoverBot Control (RBC) application enables remote users to receive a live video-stream from the robot, and allows them to send movement-commands to the RoverBot. The application has been implemented as a Jini service in order to exploit Jini code-mobility features [61]. A Jini service consists of a server-side back-end and a client-side *service object*. In compliance with the Jini service model, the RBC has been split between the client-side, i.e., the remote-driver side, and the back-end side, i.e., the Roverbot side. The overall architecture of the presented case-study application is depicted in Figure 5.6.

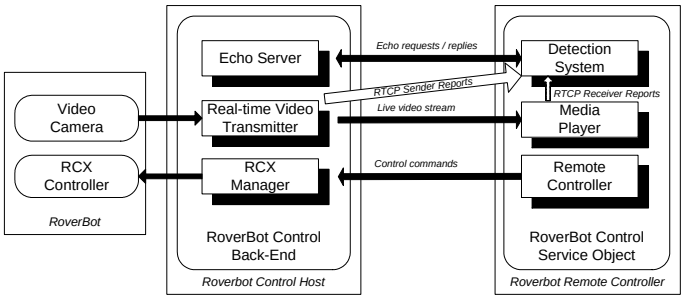

Figure 5.6: Overall architecture of the case-study application

RBC's back-end consists of the following components: *i*) the *real-time video transmitter* which produces a live video stream representing RoverBot's viewpoint; such a stream is based on the real-time transport protocol (RTP/RTCP) [54]; *ii*) the *RCX manager*, which interfaces RBC back-end with RoverBot's RCX controller, in order to force it to perform specific movements; and *iii*) the *echo server*, which is in charge of answering to echo requests sent by the *service object* for estimating the actual round trip time.

RBC's service object is composed of *i*) the *media player*, which is in charge of presenting the video stream on remote driver's display, *ii*) the *remote controller*, which allows the driver to communicate with the RBC back-end in order to move the RoverBot, and *iii*) the *detection system*, which implements the mechanism described in Section 5.1.4 with respect to an RTP-based streaming session.

Multimedia-related components and part of the detection system have been here implemented exploiting the Java Media Framework (JMF) libraries [58]. The *RCX manager* is based on the RCX Java API [35] for communicating with the RoverBot.

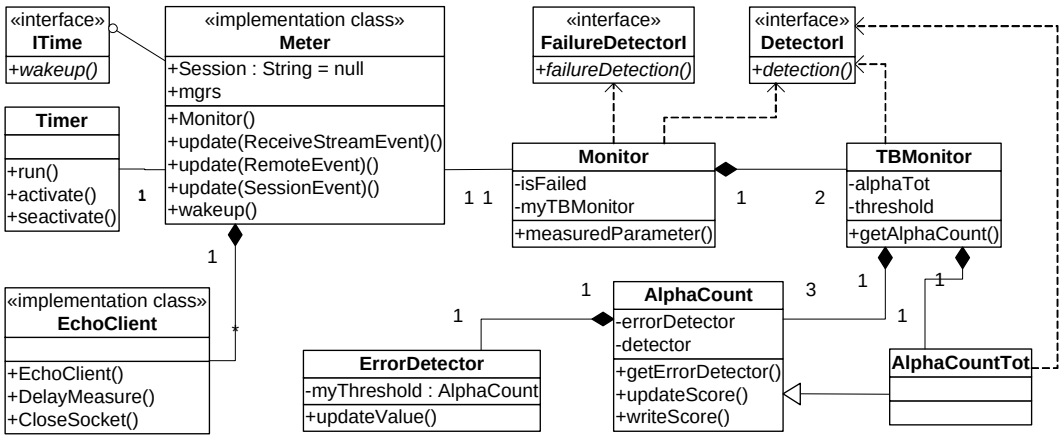

Figure 5.7: UML class diagram of the implemented service-element

Implementation of the detection service-element

The proposed service-element has been implemented as a single centralized component. The issues concerning a distributed implementation of such a component are not addressed here. However, the interested reader may refer to [51] for a thorough description of a distributed implementation of threshold-based diagnostic mechanisms. Figure 5.7 shows the class diagram of the implemented detection system.

The *meter* class is responsible for estimating delay, delay variation and information loss. Information about the quality of the actual multimedia sessions is extracted from RTCP session messages [54], called RTCP reports, which are periodically exchanged between senders and receivers of an RTP stream. The observation period varies from 0 to 5 *secs*. Such messages provide measures about *i*) the *interarrival jitter*, i.e., an estimation of the statistical variance of RTP data packets inter-arrival time, and *ii*) the *fraction lost*, i.e., the fraction of lost packets since the last report. The implemented *meter* uses such information to measure delay variation and information

loss. Since the *interarrival jitter* is measured in $msec \times SamplingTime$, delay variation is calculated by dividing the *interarrival jitter* by the bit-rate (coding/decoding frequency). RTCP reports provide no information suitable for estimating the delay. To this aim the *Meter* has been enriched by the *EchoClient* component, which is in charge of calculating the round trip time (RTT) between the remote streaming host and the client. Such component relies on the presence of the *EchoServer* on the remote host to estimate the delay between the two components. The experiments on the implemented prototype showed that the accuracy of delay measurements depends on the size of RTP packets. In order to improve the accuracy of such an estimation, the size of several RTP packets has been measured for a large variety of formats. Results are shown in Figure 5.8.

During the start-up phase, the *EchoClient* retrieves the format of the media content by analyzing RTCP reports, thus adapting the size of echo packets to the effective size of RTP ones. Each periodic evaluation of the RTT results from the mean value calculated on multiple measures; evaluations are periodically executed according to the wake-up messages provided by the *timer* thread. The value of the wake-up period was set to that of the observation.

Audio/Video	Format/Frequency[*Hz*]	Packet size [*byte*]	# Packets/sec. (*on average*)
Audio	**DVI/8000**	284	16
Audio	**DVI/11025**	374	16
Audio	**DVI/22050**	704	16
Audio	**ULAW/8000**	520	16
Audio	**GSM/8000**	139	16
Audio	**MPEG Layer III**	1298 /1141	12
Video	**JPEG/90000**	1008	(352x288) 220
Video	**H263/90000**	550 (*on average*)	(352x288) 70
Video	**MPEG-I/90000**	1000 (*on average*)	(352x288) 80

Figure 5.8: RTP packet sizes as a function of media formats

The *Monitor* class is in charge of triggering a failure signal upon failure detection. Such a class instantiates two or more *TBMonitor* objects. Indeed, a multimedia content is composed by multiple flows (e.g., the audio and video flows). The *TBMonitor* class implements the error filtering subsystem, i.e., it is in charge of detecting failures for the assigned flow. The *AlphaCount* class implements the alpha-count function. According to the conceptual model, described in section 5.1.4, the *TBMonitor* is composed of 4 *AlphaCounts*. In particular, the alpha-count which implements the TBF_{trp} function, called *AlphaCountTot*, is a specialization of the *AlphaCount*.

5.1.6 Experimental results

In real-world scenarios, quality parameters are measured during a certain observation period T. Indeed, measuring these parameters for each packet leads to unfeasible solutions due to the large measurement-overhead. For this reason the monitoring functions of the RTP protocol are based on an average observation period of $1-2[sec]$. If a permanent error on a single parameter occurs, the detection time can be estimated as follows: $dt[sec] = \frac{\alpha}{K} \times T$.

The implemented case-study application has been used to analyze and to evaluate the behavior of the proposed service-element. Experiments have been performed on a testbed, composed of a remote-control workstation (RCW), and a control host (CH) which effectively resides on a Lego-Mindstorm RoverBot, distributed over an IEEE 802.11b wireless LAN.

The RCW is a uniprocessor Linux machine running Linux Red Hat 8.0 distribution. The CH is an HP Tablet PC, which is IEEE 802.11b compliant and it is equipped with a video camera. The considered control application has the following QoS requirements: *i*) $d_{spec} = 80\ ms$, *ii*) $dv_{spec} = 10\ ms$, *iii*) il_{spec}= 3%.

The aim of the herewith presented experimental evaluation is twofold. The first goal is to validate considerations and tuning rules given in Section 5.1.4. More specifically, given the specific QoS requirements, the suitability of variable bit-rate (VBR) and constant bit-rate (CBR) encodings has been investigated. Both strategies have been tested with respect to two kinds of video images, captured by a fixed video camera and by a moving one. The former is typical of telemedicine and survellaince applications, whereas the latter is suitable for unmanned vehicle control applications. The VBR and CBR streams used the H263 and JPEG protocols respectively; the video-resolution was set to 352x288 pixels. Measures have been performed with a period $T = 2s$; hence each measure refers to 140 packets if the H263 protocol is used, whereas each JPEG-related measure refers to 440 packets, as Figure 5.8 shows. Experimental results showed that delay variation is significantly influenced by i) the kind of images, i.e., coming from fixed camera or with rapid scene changes, and ii) the encoding strategy, i.e., VBR or CBR. In particular, the H263 protocol produced a higher average delay variation, compared to that of the JPEG stream: for a fixed camera, $t^{fix}_{dvH263} = 3.73msec$ and $t^{fix}_{dvJPEG} = 0.73msec$. Moreover, upon rapid scene-changes the obtained values were $t^{mov}_{dvH263} = 9.0msec$ and $t^{mov}_{dvJPEG} = 0.88msec$. It is worth pointing out that delay variation in JPEG CBR encoding does not significantly depend on the kind of images; hence, upon rapid scene-changes, the resulting video is quite distorted. This confirms what was claimed by Section 5.1.4.

Secondly, the suitability of the implemented mechanism to detect the failure modes defined in Section 5.1.3 has been evaluated as well. Although it has been used for a critical application, the considered multimedia service can tolerate degradation of user-related parameters, until the perceived QoS allows the driver to control the

remote device. Hence, deciding whether a degradation leads to a failure or not is a challenging issue. Such an issue has been addressed by a qualitative evaluation of the received stream. Experiments described in the following validate the suitability of threshold-based filtering for distinguishing acceptable degradations from service failures. It is worth mentioning that the analysis did not take into account delay failures; indeed, experiments showed that delay values were much lower than service requirements, whereas delay variation, information loss and transparent failures have been injected. Errors e_{DV} were generated by introducing a CPU-bound load on the control host. This load resulted in an increase of the delay variation, thus forcing jitter errors to occur. The second have been forced by injecting link faults on the ethernet switch that resides between the wireless LAN access point and the RCW. To this aim the service-element has been enriched by a component for enabling and disabling switch's port at run-time through a serial cable. In order to inject transparent failures, multiple light degradations of each parameter were forced. More specifically, several degradations were transparently injected by means of the Nistnet network emulator. Figure 5.9 depicts the errors and failures detected during about 700 seconds – i.e. 350 measures – of live streaming captured by the camera on the RoverBot. The detector raised a single transparent failure signal, due to the errors detected between measures 330 and 350. Experimental results show that only significant degradations of user-related parameters resulted in a transparent failure. The behavior of the failure detector was consistent with the empirical evaluation of the received images. The device was controllable until the transparent failure was raised, although several errors were detected until measure 330. Subsequently, the RoverBot became uncontrollable. Crash failures are detected by the TBf_d functions.

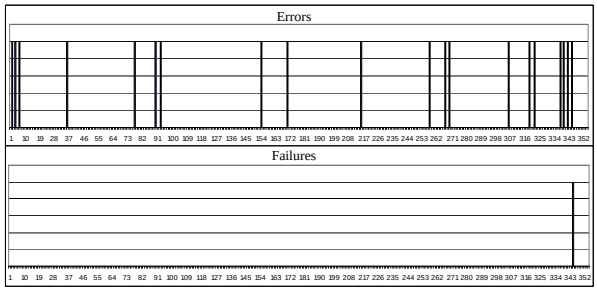

Figure 5.9: Detected errors and related failures

5.2 Supporting Location-awareness on Personal Devices

5.2.1 Location-awareness

The proliferation of handheld mobile devices and wireless networks is laying the groundwork to the development of novel context-aware systems. The context-aware applications that are based on a location-dependent context model are of special interest for nomadic computing system developers and designers. Indeed, during the last years a plenty of works presented systems and technologies for automatic location-sensing of either people or devices [1, 3, 17, 24, 26, 27, 33]; these works demonstrated the importance of providing location-sensing components to applications for personal devices (e.g., PDAs, Laptops, Mobile Phones); the main limitation of the existing approaches is that each solution solves a different problem or supports different applications.

Many mobile devices (e.g. smart-phones and PDAs) now support the Mobile Information Device Profile (MIDP) of the Java 2 Micro Edition (J2ME) platform. This platform provides a common yet flexible computing and communication environment that could be fitted for devices having different capabilities. Moreover, currently

most of mobile devices have a wireless network device such as IEEE 802.11 and Bluetooth adapters. Since in future mobile systems several technologies will coexist, such systems will need a high-level Application Programming Interface for technology-independent location sensing [43]. For this reason, generic service-elements will function as the key functional blocks for location-aware nomadic applications.

As for location-aware computing, no single location-sensing technology is likely to become dominant, since each technology can either satisfy different requirements or suit different devices. In order to deal with this heterogeneity, the Java Community Process (JCP) has recently finalized a Java Specification Request (JSR) to define a Location API (JSR-179) [47] for MIDP-compliant devices. This package provides applications with functionality for *i*) obtaining information about location and orientation of the mobile device, and *ii*) accessing a shared database of known locations (the so-called Landmarks). These specifications may be implemented by means of existing location methods, including satellite based methods like GPS, as well as short-range positioning methods.

Though the Received Signal Strength (RSS) has been used as a good location-fingerprint by several indoor positioning techniques [25, 33], no standard approach to tailor such techniques to the JSR-179 API has been proposed yet. This section proposes an implementation of these specifications, and demonstrates its effectiveness by providing experimental results obtained from a bluetooth-based system.

Specifically, this section shows how to extend the Java APIs for Bluetooth (JSR-82) [46] in order to provide the Location API with a generic service-element for RSS-based location-sensing. The proposed service-element relies on the insertion of a new component, called `RSSI_Provider`, into the JSR-82 API. This is in charge

of producing information about signal strength, which could be needed to build `Location` objects. The bridge between the the two APIs is built through a specific `LocationProvider` class (i.e. the effective location-sensing element, generating `Location`s, defined in JSR-179), which exploits the `RSSI_Provider` to generate `Location`s. The proposed solution can suit a wide variety of mobile devices, since it requires no additional positioning device but a bluetooth adapter. Based on the designed extensions, this section proposes an implementation of the `LocationProvider`, which relies on a specific RSS-based indoor positioning technique [12].

5.2.2 The Location API for J2ME CLDC profile

The JSR-179 [47] defines a J2ME optional package to enable location-aware applications for MIDP devices. Specifically this package provides the following two main functionalities: i) obtaining information about location and orientation of the mobile device; and ii) accessing a landmark database stored on the device. Each functionality exploits specific objects as information containers; indeed, the `Location` class represents the standard set of basic device-location information, whereas the `Landmark` class represents known locations (i.e. the places that mobile users can go through). `Landmark` objects have a name and may be placed either into a single category or into several categories. Each category is intended to group landmarks that are of similar type to the end user (e.g. restaurants, museums). Landmarks are stored into a persistent repository, called `LandmarkStore`, which provides JSR-179-compliant applications with a shared database for location description. The `LocationProvider` class represents a module that is able to determine the location of the terminal. This may be implemented by using existing location methods, including satellite based methods

like GPS, and short-range positioning methods like Bluetooth Local Positioning. Actually, each device can have several location providers installed, each related to a different positioning technique (e.g., GPS and RSS-based triangulation). The API allows to specify selection criteria to choose the most suitable `LocationProvider`. Upon the selection of a specific `LocationProvider`, the application can retrieve `Location` objects by means of either periodic updates or asynchronous queries.

5.2.3 Enabling RSSI-based positioning for JSR-179 compliant applications

Location API for indoor positioning

JSR-179, as already mentioned, enables developers to write wireless location-based application and services for mobile devices by means of several location methods, including satellite based methods, as well as short-range positioning methods. The accuracy of information provided through this API depends on the accuracy of the adopted positioning system.

This section refers to an indoor-positioning technique which is suitable for detecting the symbolic position [26] of mobile users within a set of buildings. For the sake of clarity, in the rest of this sub-section this technique is briefly presented. The interested reader may refer to [12] for further details on the adopted approach. It is worth pointing out that the mentioned technique aims to identify the room that the device is moving in; thus the accuracy contained into `Location` objects is related to the coordinates-error probability (i.e. the probability of associating a device to a wrong room). Since an effective measurement of such an error probability goes beyond the scope of this work, the coordinates are herewith assumed to have a constant accuracy. According to the adopted technique, each room of the building is represented

by a symbolic location, called "zone"; each zone is assigned to a certain Bluetooth position-sensor. The topology of the considered environment is represented by the *Sensor-Coordinates* table, which associates each Bluetooth sensor to a specific room, and the *Sensor-Neighbors* table, which logically links sensors each other between adjacent rooms.

The adopted technique uses the Received Signal Strength Indicator (RSSI) to finger-print the rooms that the mobile device is in. It is worth noting that, although symbolic representation of locations implies an additional layer of indirection to get the geometric information, it brings some benefits to the proposed solution. First, storage and retrieval of symbolic location data might be more suitable to location applications than geometric location data. Second, a hierarchical (e.g., building-floor-room) symbolic data model can ease the integration of multiple wireless technologies within the same application. The third advantage is that the symbolic model might be used to predict user location, for it helps building secondary models of location information such as individual mobility patterns.

The rest of this sub-section shows how the adopted positioning technique can be mapped onto the JSR-179 API. Moreover, it also shows how to extend the JSR-82 API so as to bridge it to JSR-179 `LocationProviders`, and provide some comments about the current implementation.

According to the zoning approach, each `Location` has a one-to-one relationship with a specific room of the building. Within the JSR-179 specifications, `Location` objects represent the up-to-date location in terms of timestamped coordinates, accuracy, speed, course, and information about the positioning method used for the location, plus an optional textual address. Thus, it is crucial that such a technique

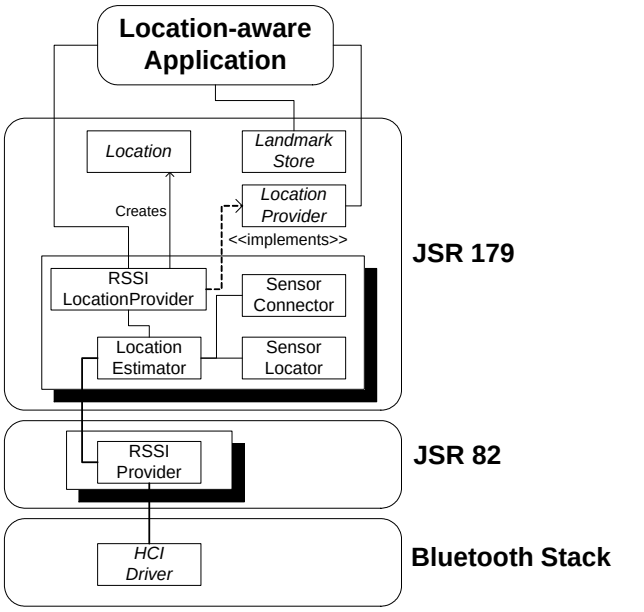

Figure 5.10: High-level architecture of the proposed service-element. Shadowed boxes contain the implemented components, whereas non-shadowed boxes represent off-the-shelf components.

be mapped onto JSR-179 semantics. Therefore, each room is assigned a specific set of latitude, longitude, and altitude parameters. In addition, as already mentioned, the accuracy is assumed to be constant. `Landmark`s and the already mentioned tables represent the topology of the considered environment. Figure 5.10 shows the overall architecture of the proposed extensions. A particular `LocationProvider`, namely the `RSSILocationProvider`, has been implemented in order to provide the overlying JSR-179 applications with the indoor location-sensing service-element. As for `Landmark`s, each known `Location` (i.e., each room) is assigned a name that identifies the room to the end user (e.g., the Contemporary Art gallery within an exhibition). As for the topology tables, the `SensorLocator` component is in charge of managing the *Sensor-Coordinates* table; hence it associates each zone (i.e. each Landmark) to a specific Bluetooth sensor. The `SensorConnector` manages the *Sensor-Neighbors* table; hence

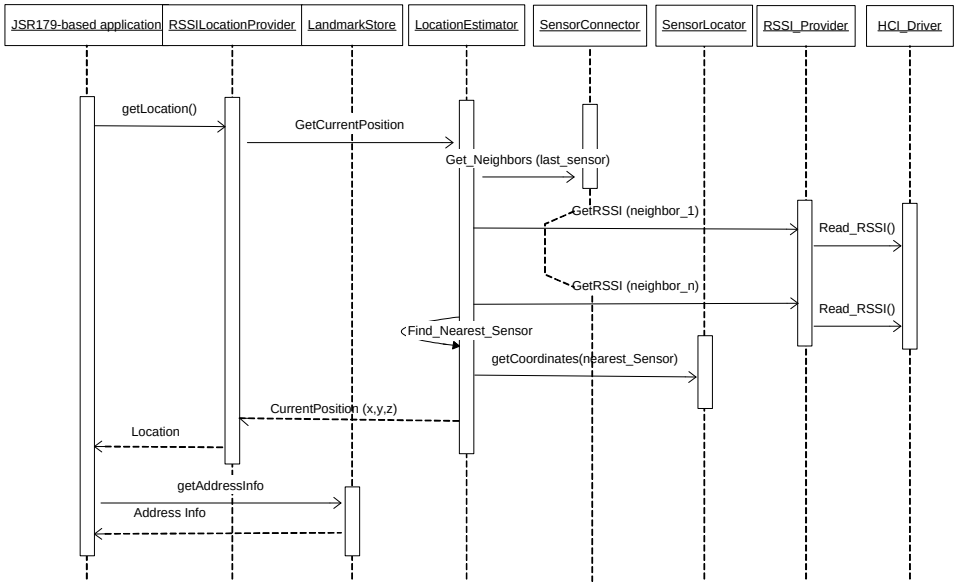

Figure 5.11: Retrieval of a mobile user's location

it represents the interconnections between rooms.

5.2.4 Enabling the RSSI-based positioning technique on a Bluetooth network

In order to enable the proposed technique on a Bluetooth-based system, a new class has been appended to the Java API for Bluetooth (JSR-82) [46], namely the `RSSI_Provider`, which allows to measure the RSSI; this value refers to the communication between the device itself and another Bluetooth device (i.e., a sensor).

Figure 5.11 shows the retrieval of user's location by means of the described extensions. It is worth noting that the `LocationEstimator` component exploits the `RSSI_Provider` to discover the zone where the mobile device is. It specifically analyzes the *Sensor-Neighbors* table by means of the `SensorConnector` component, and reads the RSSI for each neighbored sensor. Upon the identification of the nearest sensor (i.e., the identification of the current room), the `LocationEstimator` can i)

retrieve its spatial coordinates by means of the `SensorLocator` component, and *ii*) return them to the `RSSILocationProvider`.

Implementation details

The `RSSI_Provider` class has been implemented as an extension to the JBlueZ library. JBlueZ is an implementation of the JSR-82 specifications, which relies on the official Linux Bluetooth protocol stack, namely Bluez [6]. Since BlueZ is implemented as library of native procedures written in C, the `RSSI_Provider` uses the Java Native Interface (JNI) to retrieve RSSI and quality information through the BlueZ HCI layer. As for the Location API, a JSR-179 implementation skeleton consisting of a `LandmarkStore` and a `LocationProvider` has been implemented.

5.2.5 Experimental results

In order to evaluate the behavior of the designed service-element, it has been tested on Compaq iPAQ 3970 PDAs running the Familiar 0.7.0 Linux distribution. A set of ANYCOM Bluetooth dongles configured to accept connections from mobile devices function as Bluetooth Room sensors. The proposed solution has been evaluated from different view-points.

First, the RSSI has been measured while moving through a simple sequence of three adjacent rooms, namely the Green Room, the Blue Room, and the Red room. This preliminary experiment aimed to verify that the RSSI could be used as a room-fingerprint. Figure 5.12a shows the measured values of the RSSI with respect to each room's sensor. Figure 5.12b shows the environment used to perform this experiment. Since our testing user starts moving from green-room sensor's location (as Figure 5.12b shows), the distance from the green sensor functions as a location

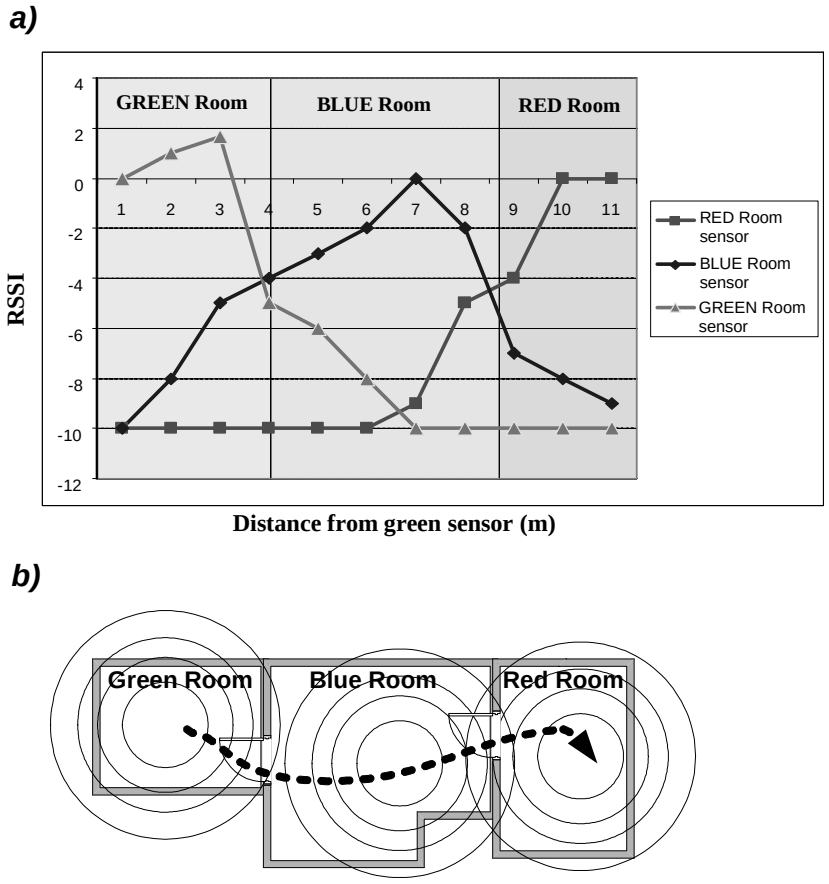

Figure 5.12: a) Signal strength while moving between different rooms; b) Topology of the RSSI-measurement environment.

variable for the presented results; the vertical dividing lines represent the separation between adjacent rooms (e.g., the wall between the green and the blue room is 4 meters far from the green room sensor). Preliminary results showed that the highest RSSI value always represents the current room: for instance, when the user is 3 meters far from the green sensor (i.e., he is walking in green room) Green Room's RSSI is higher than the others, whereas when the user is 6 meters far from the green sensor (i.e., he is walking in blue room) Blue Room's RSSI is higher than the others.

The experiments aimed also to evaluate the performance of the `RSSI_provider`

component. Specifically the RSSI measurement time $t_{Rm} = t_{BC} + t_{RSSI}$ has been measured, where t_{BC} is the time to connect to a Bluetooth sensor, and t_{RSSI} is the time to read the RSSI. Several factors influence t_{Rm}, such as channel-degradation phenomena (e.g., multi-path fading, interference, direct sunlight, physical obstacles), as well as the adopted Bluetooth hardware and software libraries. Since the measures of this preliminary evaluation focused on the performance of the implemented APIs, channel-degradation phenomena were not taken into account.

Each RSSI measurement refers to a certain sensor. Preliminary experiments showed that t_{Rm} is not influenced by the distance d from the used sensor. Indeed, it is well-known that the higher the distance between bluetooth devices is, the lower the performance of the bluetooth channel in terms of data-rate (due to packet-errors and error-correction procedures); since the amount of information exchanged between sensing devices by using the proposed protocol is extremely small, such a data-rate degradation does not significantly influence t_{Rm}. Specifically these experiments showed that $t_{BC} >> t_{RSSI}$, being $t_{BC} \approx 1175ms$ and $t_{RSSI} \approx 8ms$ independently of d.

Figure 5.13 depicts an example topology in which each room can have at most 3 adjacent rooms. According to the proposed approach, when the mobile user is walking through room Z_4, the `LocationEstimator` component continuously analyzes each adjacent rooms' sensor RSSI in order to decide whether the user has moved to a new room or not. Thus, when the user moves from Z_4 to Z_1, the Location Change Detection time t_{LCDT} depends on t_{RSSI}, whereas t_{BC} does not affect t_{LCDT} since the connection with the involved room sensors has been previously established. In this case (which is the worst case as each room can have at most 3 adjacent rooms),

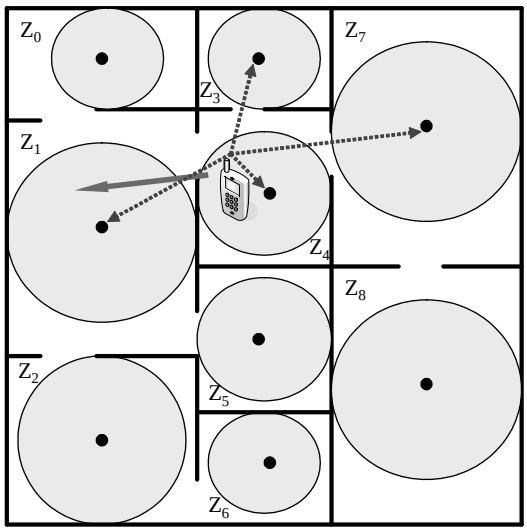

Figure 5.13: Topology of the environment used to compare the Location Change Detection Time with the speed of a walking user.

the performance of the `LocationProvider` component is suitable for the up-standing JSR-179 applications, as it is able to detect location-changes in less than 50ms; the `LocationEstimator` must compare the RSSI obtained from the current room's sensor with the value retrieved from the three adjacent rooms' sensors. Hence, in this case $t_{LCDT} \approx (t_{RSSI}(AdjRooms) + t_{RSSI}(CurrRoom)) = (3+1) \times t_{RSSI} \approx 40ms < 50ms$), which is a reasonable time if compared with the speed of a walking user (i.e., $2m \cdot s^{-1}$).

Chapter 6

Conclusions

6.1 Conclusions

The aim of this dissertation was to bring a contribution to research on nomadic computing systems with respect to the issues that arise when a high level of interoperability must be guaranteed. The ultimate goal was to enable the definition of novel strategies to support the extreme heterogeneity of future nomadic computing systems.

The overall dissertation started with a preliminary analysis of the requirements of nomadic computing systems, which allowed to identify also the most challenging issues for the realization of highly interoperable nomadic computing systems. Such a requirement analysis was followed by a critical evaluation of the state-of-the-art research in supporting nomadic computing by means of service platforms. On the one hand the state-of-the-art research in this field emphasized the unfeasible nature of a brand-new solution for addressing all the the requirements of future nomadic environments, while on the other hand it exacerbated the need for interworking in current and future nomadic computing environments. Indeed, since the requirements

of each personal computing domain are different from those of the others, the diversity of current technologies is the key factor to suit all the domains.

Building a comprehensive interworking solutions for interoperable nomadic systems is really challenging, since as the heterogeneity and the number of the involved platforms increases, a number of issues arise, including integration issues (interoperability and scalability), and unpredictability issues (context-awareness and unpredictable system behaviour).

Driven by the need for interworking, this work proposed a novel architectural framework for interoperable nomadic computing systems. The basic idea upon which the framework is based is that integration and unpredictability require two different approaches to be combined. Integration specifically requires the framework to define an innovative infrastructure for the interworking of existing service discovery and delivery technologies. However, such an approach is not suitable for addressing unpredictability issues, since such issues are highly dependent on application requirements. Hence, the framework allows to deal with such issues by means of an additional generic service-elements layer, which provides specific applications with a specialized support for addressing unpredictability.

This work showed that such a combined approach is feasible, and provided useful lessons learned and design guidelines. A proof of concept implementation of the architectural framework has also been presented, in order to point out also the main implementation issues and the strategies adopted to address them.

As for the interworking infrastructure, the implemented prototype showed that the proposed solution supports the integration of diverse Nomadic Computing Domains.

The overall infrastructure has been designed to allow interoperability of existing interaction platforms by means of two different types of agents. An innovative algorithm for proactive service advertisement has also been proposed. The effect of several filtering parameters on the import ratio (i.e., on mechanism stability) have been evaluated by simulations. Simulation results showed that system stability is proportional to the inter-filtering time: the lower the inter-filtering time is, the higher the import ratio. Moreover, the selectivity of the filtering mechanism is proportional to the number of requested operations. The conceptual infrastructure has been implemented with respect to Jini and Bluetooth technology. A proof-of-concept implementation of the overall infrastructure has been described and tested to add interoperability to an existing Bluetooth service, namely a Bluetooth printer.

As for the generic service elements, two novel techniques to handle the nomadicity concerns of location-awareness and unpredictable system behaviour have been proposed and thoroughly discussed. The first service element allowed to deal with unpredictability in RTP-based distributed multimedia applications. The second service element was designed to allow Java-based Location-aware applications to sense the location of a mobile user who is moving within a building. As for the unpredictability of multimedia applications, the proposed service-element relies on the assumption that the unpredictable behaviour of a multimedia stream can significantly affect the dependability of the applications that will use it. The key-idea upon which the service-element was built is to define the correctness of a multimedia service as a function of user-related quality parameters. The implemented prototype showed that the service element is suitable for coping with dependability issues of nomadic multimedia applications. Moreover, since the implemented prototype refers to RTP-based

multimedia streams, it can be also used to build failure-detection service-elements for other types of services that use the RTP protocol as a real-time transport layer. As for location-awareness, the service-element resulted in a versatile infrastructure model, which is independent of the used devices as well as of the wireless communication technologies. Since the solution is based on the standard Java Location API specifications (JSR-179), it is suitable for a wide range of commercial mobile devices. Experiments showed that the performance of the implemented service-element does not depend on the distance between the mobile device and location-sensors. Moreover, the performance of the implemented JSR-179 components is reasonable if compared with the velocity of walking users.

The overall work carried out during my PhD activity refers to standard technologies and interaction platform. Moreover, the description of the conceptual architecture is clearly separated from the case-study implementations. This increases the chances to provide current research with an useful and re-usable contribution, and with more general design and implementation guidelines as well.

Bibliography

[1] G. Anastasi, R. Bandelloni, M. Conti, F. Delmastro, and E. Gregoriand G. Mainetto, *Experimenting an indoor bluetooth-based positioning service*, Proceedings of the 23^{rd} International Conference on Distributed Computing Systems Workshops (ICDCSW'03), IEEE Computer Society, 2003, pp. 480–483.

[2] R. Bagrodia, W. Chu, L. Kleinrock, and G. Popek, *Vision, Issues, and Architecture for Nomadic Computing*, IEEE Personal Communications **2** (1995), no. 6, 14–27.

[3] P. Bahl and V. N. Padmanabhana, *Radar: An in-building rf-based user location and tracking system*, Proceedings of the IEEE Infocom 2000, vol. 2, 2000, pp. 775–784.

[4] John Barton and Tim Kindberg, *The challenges and opportunities of integrating the physical world and networked systems*, HP-Labs Technical report HPL-2001-18 (2001).

[5] Bluetooth SIG, *Specification of the bluetooth system - core and profiles v. 1.1*, 2001.

[6] Bluez, *Bluez: the official linux bluetooth protocol stack*, 2002.

[7] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, and F. Grandoni, *Threshold-based mechanisms to discriminate transient from intermittent faults*, IEEE Transactions on Computers **49** (2000), no. 3, 230 – 245.

[8] Wei Chen, Sam Toueg, and Marcos Kawazoe Aguilera, *On the quality of service of failure detectors*, IEEE Trans. Comput. **51** (2002), no. 5, 561–580.

[9] L. Choonhwa, A. Helal, N. Desai, V. Verma, and B. Arslan, *Konark: A system and protocols for device independent, peer-to-peer discovery and delivery of mobile services*, IEEE Transactions on Systems, Man and Cybernetics **33** (2003), no. 6, 682–696.

[10] UDDI Spec Technical Committee, *Uddi version 3.0.1: Uddi spec technical committee specification*, 2003, available at http://uddi.org/pubs/uddi-v3.0.1-20031014.pdf.

[11] The Salutation Consortium, *Salutation architecture specification, version 2.1*, June 2000, available at http://www.salutation.org.

[12] Domenico Cotroneo, Stefano Russo, Fabio Cornevilli, Massimo Ficco, and Vincenzo Vecchio, *Implementing positioning services over an ubiquitous infrastructure*, Proceedings of the Second IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (WSTFEUS'04), IEEE Computer Society, 2004, pp. 14–18.

[13] Christopher Dabrowski, William Majurski, Wayne McCoy, and Shukri Wakid, *Modeling nii services: future needs for standards and interoperability*, StandardView **2** (1994), no. 4, 203–217.

[14] Kai-Oliver Detken, Ioannis Fikouras, and Panos Phillipopoulos, *Service discovery integrated network platform*, Converged Networking: Data and Real-time

Communications over IP , Springer-Verlag IFIP International Federation for Information Processing Series **119** (2003).

[15] Myra Dideles, *Bluetooth: a technical overview*, Crossroads **9** (2003), no. 4, 11–18.

[16] UPNP Forum, *Universal plug and play device architecture, version 1.0*, June 2001, available at http://www.upnp.org.

[17] Y. Fukuju, M. Minami, H. Morikawa, and T. Aoyama, *Dolphin: An autonomous indoor positioning system in ubiquitous computing environment*, Proceedings of the IEEE Workshop on Software Technologies for Future Embedded Systems, IEEE Computer Society, 2003, pp. 53–56.

[18] Kurt Geihs, *Middleware challenges ahead*, IEEE Computer **34** (2001), no. 6, 24–31.

[19] N. Golmie, R. E. Van Dyck, A. Soltanian, A. Tonnerre, and O. Rebala, *Interference evaluation of bluetooth and ieee 802.11b systems*, Wirel. Netw. **9** (2003), no. 3, 201–211.

[20] FiCom Location API Working Group, *Ficom location api 2.0.0 interface specification*, 2002.

[21] E. Guttman and J. Kempf, *Automatic discovery of thin servers: SLP, Jini and the SLP-Jini bridge*, Proceedings of the 25th Annual Conference of the IEEE Industrial Electronics Society (1999), 722–727.

[22] E. Guttman, C. Perkins, J. Veizades, and M. Day, *Service location protocol, version 2*, IETF RFCC 2608 (1999).

[23] Uwe Hansmann, T. Stober, L. Merk, and M. Nicklous, *Pervasive computing*, Springer-Verlag New York, Inc., 2003.

[24] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster, *The anatomy of a context-aware application*, Proceedings of 5^{th} Annual International Conference on Mobile Computing and Networking (Mobicom 99), ACM Press, 1999, pp. 59–68.

[25] Jeffrey Hightower and Gaetano Borriello, *Location sensing techniques*, Techical Report UW-CSE-01-07-01, 2001, University of Washington.

[26] ———, *Location systems for ubiquitous computing*, Computer **34** (2001), no. 8, 57–66.

[27] Ekahau Inc., *Ekahau positioning engine 2.0: Developer guide*, 2002.

[28] Open Server Gateway Initiative, *Osgi service platform release 3 specification*, April 2003.

[29] Michael Jeronimo and Jack Weast, *Upnp design by example*, Intel Press, 2004.

[30] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, John Schettino, Bill Serra, and Mirjana Spasojevic, *People, places, things: web presence for the real world*, Mob. Netw. Appl. **7** (2002), no. 5, 365–376.

[31] Leonard Kleinrock, *Nomadicity: Anytime, Anywhere in a disconnected world*, Mobile Networks and Applications **1** (1996), no. 1, 351 – 357.

[32] Teemu Koponen and Teemupekka Virtanen, *A service discovery: A service broker approach*, Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9, IEEE Computer Society, 2004, p. 90284.2.

[33] A. Kotanen, M. Hannikainen, H. Lappakoski, and T.D. Hamalainen, *Experiments on local positioning with bluetooth*, Proceedings of the International Conference

on Information Technology, Computers and Communication (ITCC03), IEEE Computer Society, 2003, pp. 297–303.

[34] M.P. LaMonte, Y. Xiao, C. Mackenzie, P. Hu, W. Gaasch, J. Cullen, and D. Gagliano, *Telebat: Mobile telemedicine for the brain attack team*, J Stroke Cerebrovasc Dis. **9** (2000), 128–135.

[35] D. Laverde, G. Ferrari, and J. Stuber, *Programming lego mindstorms with java*, Syngress Publishing, 2002.

[36] Kalle Lyytinen and Youngjin Yoo, *Introduction to the special issue on issues and challenges in ubiquitous computing*, Commun. ACM **45** (2002), no. 12, 62–65.

[37] ———, *Research commentary: The next wave of nomadic computing*, Info. Sys. Research **13** (2002), no. 4, 377–388.

[38] B. Miller and R. Pascoe, *Mapping salutation architecture apis to bluetooth service discovery layer, version 1.0*, Bluetooth SIG White paper (1999).

[39] T. Nakajima, *Pervasive servers: A framework for creating a society of appliances*, Personal and Ubiquitous Computing **7** (2003), no. 3–4, 182–188.

[40] I. G. Niemegeers and S. M. Heemstra De Groot, *Research issues in ad-hoc distributed personal networking*, Wirel. Pers. Commun. **26** (2003), no. 2-3, 149–167.

[41] Open Mobile Alliance (OMA), *Oma service environment, approved version 1.0*, September 2004, available at http://www.openmobilealliance.org.

[42] Location Inter operability Forum, *Mobile location protocol lif ts 101 specification, version 3.0.0*, 2002.

[43] Cynthia A. Patterson, Richard R. Muntz, and Cherri M. Pancake, *Challenges in location-aware computing*, IEEE Pervasive Computing **2** (2003), no. 2, 80–89.

[44] Thomas F. La Porta, Krishan K. Sabnani, and Richard D. Gitlin, *Challenges for nomadic computing: mobility management and wireless communications*, Mob. Netw. Appl. **1** (1996), no. 1, 3–16.

[45] David Powell, *Failure mode assumptions and assumption coverage*, Proc. 22nd IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-22) (Boston, MA, USA), 1992, (Revised version available as LAAS-CNRS Research Report 91462, 1995), pp. 386–395.

[46] Java Community Process, *Java apis for bluetooth specification 1.0 final release*, 2002.

[47] ———, *Location api for j2me specification 1.0 final release*, 2003.

[48] Kimmo Raatikainen, Henrik B. Christensen, and Tatsuo Nakajima, *Application requirements for middleware for mobile and pervasive systems*, SIGMOBILE Mob. Comput. Commun. Rev. **6** (2002), no. 4, 16–24.

[49] Bhaskaran Raman, Sharad Agarwal, Yan Chen, Matthew Caesar, Weidong Cui, Per Johansson, Kevin Lai, Tal Lavian, Sridhar Machiraju, Zhuoqing Morley Mao, George Porter, Timothy Roscoe, Mukund Seshadri, Jimmy S. Shih, Keith Sklower, Lakshminarayanan Subramanian, Takashi Suzuki, Shelley Zhuang, Anthony D. Joseph, Randy H. Katz, and Ion Stoica, *The sahara model for service composition across multiple providers*, Proceedings of the First International Conference on Pervasive Computing, Springer-Verlag, 2002, pp. 1–14.

[50] Golden G. Richard, *Service advertisement and discovery: Enabling universal device cooperation*, IEEE Internet Computing **4** (2000), no. 5, 18–26.

[51] L. Romano, A. Bondavalli, S. Chiaradonna, and D. Cotroneo, *Implementation of threshold-based diagnostic mechanisms for cots-based applications*, Proceedings

of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02), IEEE Computer Society, 2002, p. 296.

[52] Daniel Salber, Anind K. Dey, and Gregory D. Abowd, *The context toolkit: aiding the development of context-enabled applications*, Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press, 1999, pp. 434–441.

[53] R. Schollmeier, *A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications*, Proceedings of the First International Conference on Peer-to-Peer Computing (P2P01), 2001, pp. 101–102.

[54] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *Rtp: A transport protocol for real-time applications, ietf request for comment (rfc1889)*, 1996, available at http://www.ietf.org/rfc/rfc1889.txt.

[55] Jean-Marc Seigneur, Gregory Biegel, and Christian Damsgaard Jensen, *P2P with JXTA-Java Pipes*, Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java, Computer Science Press, Inc., 2003, pp. 207–212.

[56] Joao Pedro Sousa and David Garlan, *Aura: an architectural framework for user mobility in ubiquitous computing environments*, Proceedings of the IFIP 17th World Computer Congress - TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture, Kluwer, B.V., 2002, pp. 29–43.

[57] Richard Staehli, Jonathan Walpole, and David Maier, *A quality-of-service specification for multimedia presentations*, Multimedia Syst. **3** (1995), no. 5-6, 251–263.

[58] Inc. Sun Microsystems, *Java media framework api ver.2.1.1e*, 2003, available at http://java.sun.com/products/java-media/jmf/.

[59] Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, and Bill Yeager, *Project jxta 2.0 super-peer virtual network*, May 2003, available at http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf.

[60] Anand Tripathi, *Challenges designing next-generation middleware systems*, Commun. ACM **45** (2002), no. 6, 39–42.

[61] Jim Waldo and Ken Arnold, *The Jini specification - second edition*, Addison-Wesley, 2000.

[62] P. Wallich, *Mindstorms: not just a kid's toy*, IEEE Spectrum **38** (2001), no. 9, 52–57.

[63] Mark Weiser, *Some computer science issues in ubiquitous computing*, SIGMOBILE Mob. Comput. Commun. Rev. **3** (1999), no. 3, 12.