Alma Mater Studiorum - Università degli Studi di Bologna

Dottorato di Ricerca in
Ingegneria Elettronica, Informatica e delle Telecomunicazioni
Ciclo XX
ING-INF/05

Semantic-based Middleware Solutions to Support Context-Aware Service Provisioning in Pervasive Environments

Dissertazione presentata
da
Alessandra Toninelli

Coordinatore del Dottorato
Chiar.mo Prof. Ing. Paolo Bassi

Relatore
Chiar.mo Prof. Ing. Maurelio Boari

Esame Finale Anno 2008

Author

Alessandra Toninelli

Thesis advisor

Chiar.mo Prof. Ing. Maurelio Boari

Title

Semantic-based Middleware Solutions to Support Context-Aware Service Provisioning in Pervasive Environments

Abstract

The dynamicity and heterogeneity that characterize pervasive environments raise new challenges in the design of mobile middleware. Pervasive environments are characterized by a significant degree of heterogeneity, variability, and dynamicity that conventional middleware solutions are not able to adequately manage. Originally designed for use in a relatively static context, such middleware systems tend to hide low-level details to provide applications with a transparent view on the underlying execution platform. In mobile environments, however, the context is extremely dynamic and cannot be managed by a priori assumptions. Novel middleware should therefore support mobile computing applications in the task of adapting their behavior to frequent changes in the execution context, that is, it should become context-aware.

In particular, this thesis has identified the following key requirements for novel context-aware middleware that existing solutions do not fulfil yet. (i) Middleware solutions should support interoperability between possibly unknown entities by providing expressive representation models that allow to describe interacting entities, their operating conditions and the surrounding world, i.e., their context, according to an unambiguous semantics. (ii) Middleware solutions should support distributed applications in the task of reconfig-

uring and adapting their behavior/results to ongoing context changes. (iii) Context-aware middleware support should be deployed on heterogeneous devices under variable operating conditions, such as different user needs, application requirements, available connectivity and device computational capabilities, as well as changing environmental conditions.

Our main claim is that the adoption of *semantic metadata* to represent context information and context-dependent adaptation strategies allows to build context-aware middleware suitable for all dynamically available portable devices. Semantic metadata provide powerful knowledge representation means to model even complex context information, and allow to perform automated reasoning to infer additional and/or more complex knowledge from available context data. In addition, we suggest that, by adopting proper configuration and deployment strategies, semantic support features can be provided to differentiated users and devices according to their specific needs and current context.

This thesis has investigated novel design guidelines and implementation options for semantic-based context-aware middleware solutions targeted to pervasive environments. These guidelines have been applied to different application areas within pervasive computing that would particularly benefit from the exploitation of context. Common to all applications is the key role of context in enabling mobile users to personalize applications based on their needs and current situation.

The main contributions of this thesis are (i) the definition of a metadata model to represent and reason about context, (ii) the definition of a model for the design and development of context-aware middleware based on semantic metadata, (iii) the design of three novel middleware architectures and the development of a prototypal implementation for each of these architectures, and (iv) the proposal of a viable approach to portability issues raised by the adoption of semantic support services in pervasive applications.

Contents

List of Figures

List of Tables

Citations to Previously Published Work

Large portions of Chapters 4, 5 and 6, as well as some of Chapter 3, have appeared in the following papers:

Alessandra Toninelli, Antonio Corradi, and Rebecca Montanari.
Semantic-based discovery to support mobile context-aware service access.
Computer Communications Journal, Special Issue on Mobility Management and Wireless Access, 31(5): 935-949. Elsevier, 2008.

Dario Bottazzi, Rebecca Montanari, and Alessandra Toninelli.
A Semantic Context-Aware Middleware Level Solution to Support Anytime and Anywhere Social Networks.
IEEE Intelligent Systems, Special Issue on Social Computing, 22(5):23-31. IEEE Computer Society Press, 2007.

Paolo Bellavista, Antonio Corradi, Rebecca Montanari, and Alessandra Toninelli.
Context-Aware Semantic Middleware for Next Generation Mobile Systems.
IEEE Communications Magazine, Special Issue on Advances in Service Platform Technologies, 44(9): 62-71, IEEE Communications Society, 2006.

Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila.
A Semantic Context-Aware Access Control Framework for Secure Collaborations in Pervasive Computing Environments.
Proceedings of the Fifth International Semantic Web Conference (ISWC), LNCS 4273: 473-486. Springer, 2006.

Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila.
Proteus: A Semantic Context-Aware Adaptive Policy Model.
Proceedings of the IEEE 2007 International Workshop on Policies for Distributed Systems and Networks (POLICY), pp. 129-140. IEEE Computer Society Press, 2007.

The complete list of publications is included in Appendix A.

Acknowledgments

I would like to express my thanks to all the people who supported my research activity during the last four years. Prof. Maurelio Boari and Prof. Antonio Corradi, for guiding my doctorate with their knowledge and experience. Prof. Rebecca Montanari, who has constantly encouraged and assisted me not only as a supervisor, but also as a friend. Prof. Paolo Bellavista, for his valuable help and collaboration.

I also would like to thank Prof. Piero Bonatti, who gave me the chance to collaborate on his interesting research activity at the University of Naples.

I am particularly grateful to Ora Lassila for sharing with me his knowledge, friendship and exciting research vision. My visit at NRC was a unique personal and professional experience, and strongly motivated me to pursue research towards the realization of the Semantic Web. Thanks to Deepali Khushraj, for the great time we had in Boston, both inside and outside the office, and to Lalana Kagal, for offering me valuable help and advice for my research- and many slices of cakes in our tea breaks at MIT, too.

My colleagues succeeded in making enjoyable every day I spent at the lab. I would like to particularly thank Carlo Giannelli who has shared with me bad and good times, including the great effort of writing this dissertation. Many thanks to all the friends who encouraged and supported me throughout these years.

Finally, I would like to express my gratitude to my family, which has supported me with love and patience: my mother, Elena and Nicola, without forgetting little Irene.

To my niece Irene - the journey begins.

Chapter 1

Introduction

1.1 Background and Motivation

During the last three decades, the increase of network and application logic complexity has significantly raised the demand for adequate middleware solutions. The term *middleware* defines the set of (reusable) support services that facilitate the design, deployment and execution of a distributed application by handling the complexity of the underlying networked system. In a distributed system, middleware is responsible for providing several functionalities to support the development and execution of applications. In particular, the following can be identified as primary objectives of a middleware infrastructure [108]:

- the interaction and/or integration of possibly heterogeneous systems (such as networks and resource management systems);
- the abstraction of the underlying facilities in a way that hides network, operating system and programming language heterogeneity;
- the implementation of these abstractions in transparent application programming interfaces (APIs)

Design and implementation choices in middleware development are essentially driven by the requirements of distributed systems the middleware is in charge of managing. As new application scenarios enabled by improved connectivity technologies and more powerful programming paradigms emerge, novel middleware solutions need to be designed and implemented that are able to manage the distributed systems supporting those scenarios. Therefore, methodological approaches to middleware design and development have been evolving over time according to the changing characteristics of distributed systems, which particularly concern two main directions:

- *scale*, i.e., the dimension and potential boundaries of the system;
- system *complexity*, in terms of both system components heterogeneity and their behavior dynamicity.

From the early 1970s, middleware solutions were developed according to the evolution of programming paradigms, from early Remote Procedure Call (RPC)-based systems, to Object Oriented (OO) systems, to Message-Oriented Middleware (MOM). These systems were typically built within intra-organization boundaries following a client-server approach, with fixed components whose behavior was completely predictable. Middleware targeted at this kind of traditional distributed systems is also referred to as *conventional middleware* [79].

Due to the increasing diffusion of distributed systems, conventional middleware solutions have proliferated. Several middleware support systems were developed, each defining specific primitives and protocols based on the underlying network platform and operating system. As soon as the need to integrate independently developed applications based on different platforms arose, novel middleware solutions were required to handle this integration. *Enterprise Application Integration* middleware solutions, such as Message Brokers and Workflow Management Systems, were thus developed to cope with the integration of exist-

ing applications rather than the creation of new applications from scratch, and to manage the increasing complexity of the integration logic.

The need to integrate, however, is not limited to the systems within a single company. Any middleware system often needs to interact with another middleware systems, where both support similar services and functionalities. Similar advantages can be obtained from intra-enterprise as from inter-enterprise application integration, or business-to-business (B2B) integration. With the widespread diffusion of the World Wide Web, middleware solutions for B2B application integration were required to provide support for the integration and deployment of loosely coupled, autonomous and independent software building blocks over a huge-scale distributed platform, i.e., the Web. Application servers and Web Services, which can be thought of as a Web-oriented implementation for a Service Oriented Architecture (SOA), represent notable examples of what we can call *Web-enabled middleware*.

With the advent of *pervasive computing*, the realm of distributed applications moved from the virtual world of the Web to the physical world, where humans live and operate. In pervasive environments, each user, equipped with a portable device, is able to access all services in any way at any time anywhere, thanks to the connectivity powered by modern network technologies [31]. Disconnections may frequently happen, either voluntarily, e.g., to save battery, or unexpectedly, e.g., due to a loss of signal. Portable devices may significantly differ one from another with respect to their computational capabilities, technical features and equipment, such as battery or screen resolution, communication abilities, e.g., supported wireless protocols, size and dimension. Users, which typically exhibit variable levels of technical expertise, might change their location at any time, even unpredictably. Therefore, pervasive environments are characterized by a significant degree of heterogeneity, variability and dynamicity that conventional middleware solutions are not

able to adequately manage. Originally designed for use in a relatively static context, such middleware systems tend to hide low-level network details to provide applications with a transparent view on the underlying execution platform. In mobile environments, though, the context is extremely dynamic and cannot be managed by *a priori* assumptions. *Mobile middleware* should therefore support mobile computing applications in the task of adapting their behavior to frequent changes in the execution context. In other words, middleware should become *context-aware* [21]. This requirement is twofold: on the one hand, the middleware layer should collect and represent context information at a high level of abstraction, and propagate its visibility up to the application level. On the other hand, it should provide powerful means to specify and enforce context-dependent adaptation strategies of the application, without interfering nor modifying the application logic.

It is worth noting that the behavior of a mobile application will adapt to the current context inasmuch as the needed context information and context-dependent behavior strategies are represented in a both correct and effective manner. Therefore, a crucial issue for the achievement of context-awareness is the ability of the middleware support to properly describe and interpret context information, such as the entities that characterize the system, the interactions occurring between them and the operating conditions under which such interactions occur, as well as context-driven adaptation directives. In addition, in order to provide openness and interoperability, the semantics of those descriptions must be unambiguously defined. Based on that information, disparate applications should be enabled by the middleware platform to dynamically interoperate with minimal human intervention. Describing system components' characteristics and behavior strategies is, however, a very demanding task.

We argue that the reason why this task is particularly difficult lies in the assumption that the conceptual model underlying system description and management is essentially

implicit, i.e., it is only known to humans who develop the middleware platform (and possibly encoded in natural language). The inability of the middleware platform to acquire and process knowledge about the system it is supposed to manage has hindered until now the achievement of seamless interoperability in pervasive environments. Web-enabled middleware solutions suffer from a similar limitation since Web technologies only allow to describe resources in a way that was primarily intended for human comprehension and exploitation.

1.2 Thesis Statement

The lack of explicit semantics in current middleware support for distributed environments has motivated us to explore new research directions towards novel, semantic-enabled middleware solutions. In particular, we have identified the following key requirements for semantic-enabled middleware that existing middleware solutions do not fulfil yet.

- *Middleware solutions should support interoperability between possibly unknown entities*, by providing expressive representation models that allow to describe interacting entities, their operating conditions and the surrounding world, i.e., their *context*, according to an unambiguous semantics. Context is a complex notion that has many definitions [91, 17]. Here we define context as any useful information to characterize an entity and the world in which this entity operates. The explicit representation of context information in a form that is automatically processable by a software component allows possibly unknown entities to dynamically establish an interaction based on the semantic information they can reciprocally exchange.

- *Middleware solutions should support distributed applications in the task of reconfiguring and adapting their behavior/results to ongoing context changes.* The middleware layer should therefore be able to collect, represent and reason about the context, and

to propagate this information up to the application level, i.e., it should provide the application with *context-awareness*. Context-aware adaptation strategies should be expressed at a high level of abstraction by cleanly separating application management from application logic. This separation of concerns is crucial to reduce the complexity of developing applications for pervasive environments and to favor rapid application prototyping, runtime configuration, and maintenance.

- *Context-aware middleware support should be deployed on heterogeneous devices under variable operating conditions*, such as different user needs and application requirements, device connectivity abilities and computational capabilities, as well as changing environmental conditions. Since access terminals used by mobile users might significantly differ in resource capabilities, such as display size and resolution, computing power, memory, network bandwidth and battery, and user application requirements cannot a priori determined, a crucial issue in the design and development of context-aware middleware solutions remains how to deploy middleware components on board of resource-constrained mobile devices to operate in changing context conditions.

We claim that the adoption of *semantic metadata* to represent context information and context-dependent adaptation strategies allows to build context-aware middleware suitable for the provisioning to all the devices dynamically available.

Semantic metadata provide powerful knowledge representation means to model even complex context information, and allow the dynamic extension of defined context models with additional concepts and properties. As a key feature, semantic languages allow the formal specification of context models whose underlying semantics is unambiguously defined, thus facilitating the dynamic exchange of context knowledge between interacting entities without loss of meaning. Semantic technologies also allow to perform automated

reasoning to infer additional and/or more complex knowledge from available context data. The ability to reason over context knowledge can be successfully exploited to build middleware solutions capable of recognizing context and taking appropriate management decisions based on current context. Finally, as far as the third outlined requirement is concerned, we suggest that, by adopting proper configuration and deployment strategies, semantic support features can be provided to differentiated users and devices according to their specific needs and current context.

In particular, this thesis has investigated novel design guidelines and implementation options for semantic-enabled middleware solutions targeted to pervasive environments.

1.3 Thesis Contribution

The main contributions of this thesis are:

- The complete definition of a model for the design and development of context-aware middleware based on semantic metadata, where with complete we mean that the model can precisely identify the types of semantic metadata required to describe the context of an application and the needed context-dependent adaptation strategies, can specify how to represent and express supported metadata, and can exploit them to propagate context-awareness up to the application level.

- The identification of significant application areas in the field of pervasive computing, where the demand for context-aware middleware support has not been fulfilled yet.

- The development of a set of context-aware middleware architecture prototypes targeted to the identified application areas. These prototypes provide an implementation of the metadata-based model for context-aware middleware, offer a wide range of mechanisms to collect and manage relevant context information, and propagate it up

to the application level. A key feature common to the developed middleware infrastructures is the exploitation of semantic technologies to represent and reason about context information.

- The proposal of a viable approach to the issue of portability raised by the adoption of semantic support services in pervasive applications.

With a finer degree of details, the thesis provides some novel contributions to research in the field of middleware for pervasive environments along different directions. A primary novel contribution is the exploitation of metadata to represent and reason about context. Metadata describe the structure and meaning of the entities composing a system, and the specification of management operations expressed at a high level of abstraction [77]. Among the different possible types of metadata, this thesis considers profiles and policies. *Profiles* represent characteristics, capabilities and requirements of system components, such as users, devices and services. *Policies* express the choices ruling system behavior, in terms of the actions subjects can/must operate upon resources [90]. Profiles and policies are maintained completely separated from system implementation details and are expressed at a high level of abstraction, thus achieving the clean separation of concerns between context-aware application management and application logic.

The effectiveness of metadata adoption depends on the characteristics of both the chosen specification language and the middleware support infrastructure. A second research contribution of this thesis is to suggest that semantic technologies represent a valid option for metadata specification and management. Semantic languages permit to explicitly represent interacting entities at a high level of abstraction, such as services, resources and users, and their context, such as current location of users/devices, state of resources, user preferences, and device characteristics, while enabling automated reasoning about this rep-

resentation. This favors the dynamic interoperability and mutual comprehension between entities sharing little or no prior knowledge about each other. In addition, the adoption of semantic languages for metadata specification simplifies metadata reuse and facilitates the analysis of potential conflicts and inconsistencies. In particular, we express semantic metadata using a Semantic Web standard language, i.e., the Web Ontology Language (OWL) [20].

Finally, related to the issue of providing resource-constrained portable devices with adequate context-aware middleware support, this thesis provides original results in applying the proposed metadata model to properly configure semantic support on mobile devices. In particular, we suggest that semantic support functionalities, mobile device properties, as well as configuration strategies needed to deploy semantic support components on mobile devices, can be represented by means of appropriate metadata. This allows to exploit the same management and adaptation mechanisms developed for context-awareness to properly configure semantic support based on mobile device properties.

1.4 Thesis Outline

This section presents the content of all remaining chapters. Chapter 2 in this thesis will provide, after a brief digression on the peculiarities of pervasive environments, an overview of the guidelines for the design, implementation and deployment of context-aware middleware solutions. The chapter will also review previous research work on context-aware and mobile middleware that is relevant for the present thesis. Chapter 3 will provide an overview of our approach to the design of semantic-enabled context-aware middleware, and will introduce some relevant application areas we have chosen in the field of pervasive computing to prove the usefulness and feasibility of our approach. Chapter from 4 to 6

will describe the semantic-enabled context-aware architectures we have designed and implemented in those application areas. It will proceed by describing the semantic metadata model and the middleware support services that compose the architecture in charge of providing a concrete deployment of the model. Each chapter from 4 to 6 will also provide implementation details of the proposed architectures, experimental results to evaluate our middleware prototypes, and an overview of related work in the specific application area. Finally, Chapter 7 will be devoted to give a critical analysis of the work, will provide the conclusions and outline future research works.

Chapter 2

Context-Aware Mobile Middleware

Metadata are data about data. Middleware is software about software.
Nick Gall

Telecommunication systems and the Internet are converging towards an integrated pervasive scenario that permits users to access resources and applications anytime, anywhere even when they are on the move. The diffusion of pervasive scenarios calls for appropriate middleware support solutions that are able to adequately support the increased complexity and dynamicity of emerging mobile applications. This chapter, after a brief introduction to pervasive environments, surveys the management issues arising in mobile and pervasive applications and the consequent requirements for mobile middleware support solutions. Then, the chapter presents those research efforts that we consider most significantly related to this thesis work. In particular, it first describes different models for context representation. Then, it presents and classifies several approaches to context information provisioning and management, and gives an overview of most relevant context-management infrastructures in the research literature. Finally, it outlines the main research directions in the design

of middleware solutions supporting context-aware applications for pervasive scenarios, and presents significant examples for each of the defined research direction.

2.1 Design Requirements for Context-Aware Middleware

Compared to conventional distributed applications, pervasive computing environments are characterized by new issues that make service provisioning a rather demanding task. Mobility of users and access devices is pushed to the extreme. Users can connect to the network from ubiquitous points of attachment and wireless portable devices can roam by maintaining continuous connectivity. Frequent disconnections of users/devices are rather common operating modes that can occur either voluntarily to reduce connection costs and to save battery power or accidentally due to the loss of wireless connectivity. In addition, pervasive scenarios exhibit a high degree of heterogeneity of both access devices, in terms of screen size/resolution, computing power, memory, operating system, and supported software, and networking technologies, e.g., IEEE 802.11b/g, Bluetooth, GSM, GPRS, and UMTS).

The distinctive features of pervasive computing pose new challenges in retrieving and operating on distributed resources and undermine several assumptions of traditional service provisioning scenarios. The main impact derives from the notion and the new meaning of *context*. Context is a complex concept that has been given several definitions [41]. Hereinafter, at a high level, the term context is defined as any information that is useful for characterizing the state or the activity of an entity or the world in which this entity operates.

Conventional middleware relies on a relatively static characterization of the context, where resource availability is independent of both the user current location and the

access device properties (location and heterogeneity transparency). Changes in the set of accessible resources are relatively small, rare, or predictable. Originally designed for use in such a static context, conventional middleware systems tend to hide low-level network details to provide applications with a transparent view on the underlying execution platform. In mobile environments, however, the context is extremely dynamic and cannot be managed by *a priori* assumptions: context variations can be very frequent, especially when using wireless portable devices.

Supporting mobile applications in pervasive environments thus requires to provide context visibility, where context is represented not only by location information but also by other system-level data, such as access device characteristics, environmental conditions, e.g., time and temperature, and available resources' state. This information should be propagated up to the application level to dynamically determine each mobile user's context and to perform application adaptation accordingly. Due to the high level of variability and heterogeneity, pervasive application management is a very complex task, which requires novel methodologies and tools to specify which management actions should be taken based on context information and to promptly carry out the desired context-dependent application adaptation. Context-aware behavior strategies should be expressed at a high level of abstraction by cleanly separating service management from service logic. This *separation of concerns* is crucial to reduce the complexity of developing services for pervasive environments and to favor rapid application prototyping, runtime configuration, and maintenance.

The above considerations call for the design of novel middleware solutions to support the context-aware adaptation of pervasive applications. In particular:

- The middleware should be designed according to a *cross-layer* approach, where application management layers interact with the underlying layers to collect relevant information for context determination, e.g., current location of users/devices, state

of resources, user preferences, and device characteristics. Such cross-layer interaction should enable the middleware to dynamically acquire, represent and process context information, and propagate it up to the application level.

- The middleware should provide powerful means to represent and enforce context-dependent adaptation strategies for mobile applications, without interfering nor modifying the application logic, according to a clean separation of concerns principle.

To address the above outlined requirements, different approaches to mobile middleware design and development have emerged in recent years. On the one hand, relevant research works have tackled the issue of providing adequate means to represent, collect and provide context information to applications. On the other hand, significant effort has been spent to design novel middleware architectures that support context visibility at a high level of abstraction and allow the context-dependent adaptation of mobile applications, while leaving the application logic intact. The following sections provide an overview of emerging solutions for context information modeling and provisioning, and illustrate significant approaches to the design of context-aware, adaptive middleware.

2.2 Context Models

Several research efforts have been directed in the last decade towards the design of suitable models for context information management. While early models mainly addressed the modeling of context with respect to one application or an application class, generic context provisioning models soon became of interest since many different applications could exploit them. First steps towards an agreed understanding of context have been taken, mostly with respect to common information such as location, identity, and time. However, the notion of context still remains subject to many different interpretations and the

currently prevailing research approach is to define the concept of context as much generic as possible, while leaving to applications the possibility to further refine the meaning of context according to their specific purposes. The main objective of current research in the area is therefore to develop uniform context representation models and query languages, as well as reasoning algorithms that facilitate context sharing and application interoperability.

2.2.1 Context Representation

The definition of an adequate model for context representation represents a crucial step in the process of designing context-aware applications. In the literature the term *context-aware* first appeared in [41], where Dey and Abowd describe context as location, identities of nearby people and objects and changes to those objects. One of the best topical definitions is due again to the same authors, who defined context as "any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves" [40]. Several alternate definitions of the term context can be found in literature. A detailed discussion of the differences between them is however out of the scope of this section. For a more comprehensive analysis of the topic we refer the reader to [91].

Throughout this section we will survey the most relevant context modeling approaches, by classifying them based on the data structure scheme used to represent and share contextual information in the systems where such models were defined.

- **Key-value pairs**. The model of key-value pairs is the simplest data structure for modeling contextual information. Schilit et al. [88] used key-value pairs to model the context by providing the value of a context information, e.g., location information, to an application as an environment variable. The key-value modeling approach is

frequently used in distributed service frameworks, such as discovery frameworks, e.g., Jini [3] or SLP [9], where service functionalities are described with a list of simple attribute-value pairs, and the discovery procedure operates an exact matching on these attributes. Similarly to service attributes, in those systems context information is described in terms of attribute-value pairs. Key-value pairs are easy to manage, but lack capabilities for sophisticated structuring to enable efficient context retrieval algorithms.

- **Markup scheme**. This approach defines a hierarchical data structure consisting of markup tags with attributes and content. In particular, the content of the markup tags is often recursively defined by other markup tags. Markup-based context representations usually exploit a derived language of the Standard Generic Markup Language (SGML) [7], the superclass of all markup languages, to serialize context information. The most commonly adopted language is the eXtensbile Markup Language (XML) or one of its vocabularies [29]. This kind of context modeling approach typically requires the specification of profiles (for a detailed discussion on profiles, see Section 2.3.1).

- **Graphical model**. A very well known general purpose modeling instrument is the Unified Modeling Language (UML) which has a strong graphical orientation (UML diagrams). Due to its generic structure, UML is also appropriate to model the context. A relevant example is the graphic-oriented context model introduced in [54] by Henricksen et al., which is a context extension of the Object-Role Modeling (ORM) approach [53] according some contextual classification and description properties. A graphical approach to context modeling is particularly suited to database-oriented applications, for example to derive Entity-Relationship (ER) context models and store them into relational databases.

- **Object-oriented model**. Object-oriented approaches to context modeling aim at exploiting the main benefits of this approach, namely encapsulation and reusability, in the process of representing and accessing context in ubiquitous environments. By relying on the abstraction of *object*, they encapsulate and hide from external access the details about context collecting and processing, while providing contextual information by means of interfaces. For example, the concept of *cues* developed within the TEA project [89] provides an abstraction for physical and logical sensors: in particular, a cue represents a function, which takes as an input the value of a single physical or logical sensor at a certain time, and provides as an output the symbolic representation of a certain context.

- **Logic-based models**. In a logic-based context model, context is represented and processed by means of facts, expressions and rules. Generally speaking, a logic defines the conditions on which a concluding expression or fact may be derived (a process known as reasoning or inferencing) from a set of other expressions or facts. To describe these conditions, a set of rules a formal system is applied. Contextual information is represented by means of logical expressions. It is added, updated and (possibly) deleted from a logic based system in terms of facts, or inferred from appropriate rules defined in the system. All logic based models share a rather high degree of formality in context representation and processing. For example, the Sensed Context Model proposed by Gray and Salber [50] exploits first-order predicate logic as a formal representation of contextual propositions and relations. Another approach within this category is the framework GAIA [83]. Other solutions adopt additional logics, such as for instance fuzzy logic, to represent and reason about uncertain context information or determine the quality of context information [24]. Let us note that the exploitation of logics in context representation allows automated reasoning over

context information.

- **Ontology-based models**. According to Gruber's definition, an ontology can be defined as "a formalization of a conceptualization" [51]. In a semantic approach, ontologies allow the description of context within specific knowledge domains by means of explicit formalisms, which can be used to represent and reason about context information. Semantic-based context models represent an emerging approach in context representation since they support knowledge sharing and reuse, and logical inferencing capabilities. Relevant examples include the CONON context modeling approach by Wang et al. [105] and the SOUPA ontology developed within the CoBrA system [34]. These approaches will be discussed in more detail in the following section, which provides some insights on Semantic Web technologies and their exploitation for context representation and reasoning.

2.2.2 Semantic Web Languages for Context Modeling

Semantic languages have gained considerable attention within the pervasive research community as a suitable means to provide expressive context representation, querying and reasoning support [34, 106, 73]. Compared to alternate representation models, semantic-based approaches are emerging because of the several advantages they bring in context modeling. Semantic languages permit to describe at a high level of abstraction the structure and properties of the entities composing a pervasive system, e.g., users, devices and resources, and the desired management operations to govern and control entity behavior. These features appear to be particularly attractive in ubiquitous environments characterized by constantly changing context conditions. The adoption of ontologies to describe context in pervasive computing scenarios brings several advantages by allowing the exchange of semantics about the described context, it enables mutual understanding between previously

unknown entities about their capabilities and the current execution context. Moreover, Semantic Web languages enable expressive querying and automated reasoning over context representation, to derive additional and/or higher level context information that can be exploited by the application.

In this section, we first provide an overview of Semantic Web standard languages, followed by a description of relevant existing work on ontology-based context representation models.

Resource Description Framework

Semantic Web technologies can be thought as a layered framework, whose lower layers provide data interchange formats, both syntactic and semantic, on top of which ontologies can be build, queried and possibly supplemented by rules, as shown in Figure 2.1[1]. In particular, the Resource Description Framework (RDF) is a language originally created for representing information about resources in the World Wide Web [66]. By generalizing the concept of a Web resource, RDF can also be used to represent information about resources that cannot be directly retrieved on the Web, including user preferences, mobile device properties and any other context information. It is worth noting that RDF provides a common framework for expressing the semantics of this information so it can be exchanged between applications without loss of meaning. RDF identifies things using Web identifiers (called Uniform Resource Identifiers, or URIs), and describes resources in terms of simple properties and property values. In particular, RDF represents simple statements about resources as a graph of nodes and arcs representing the resources, and their properties and values. For example, the statement "Dave Beckett is the editor of the resource http://www.w3.org/TR/rdf-syntax-grammar" can be represented by the graph

[1] http://www.w3.org/2007/03/sw

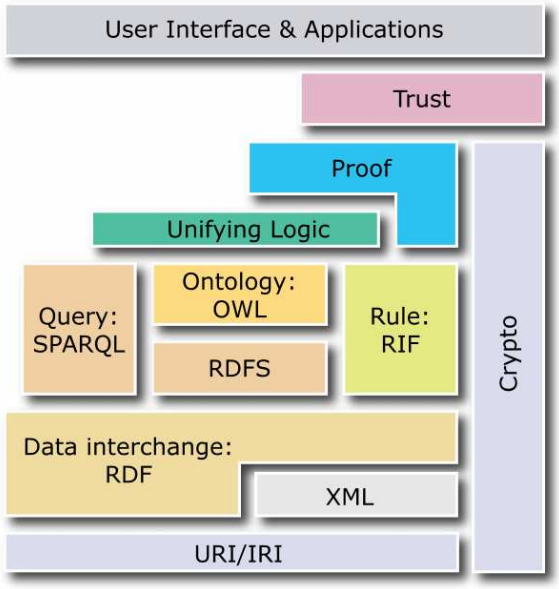

Figure 2.1: Semantic Web layered framework.

depicted in Figure 2.2[2]. To encode RDF statements in a machine-processable way, RDF relies on a serialization based on the Extensible Markup Language (XML) [29].

RDF properties may be thought as attributes of resources, but may also represent relationships between resources. RDF however, provides neither mechanisms for describing these properties, nor does provide mechanisms for describing the relationships between these properties and other resources. To overcome this limitation, some extensions such as the RDF Schema (RDF-S), i.e., the RDF vocabulary description language, have been defined [30]. RDF-S defines classes and properties that may be used to describe classes, properties and other resources. RDF-S defines classes and properties that may be used to describe classes, properties and other resources, such as the domains and ranges of properties.

[2]http://www.w3.org/TR/rdf-syntax-grammar/#section-Syntax-node-property-elements

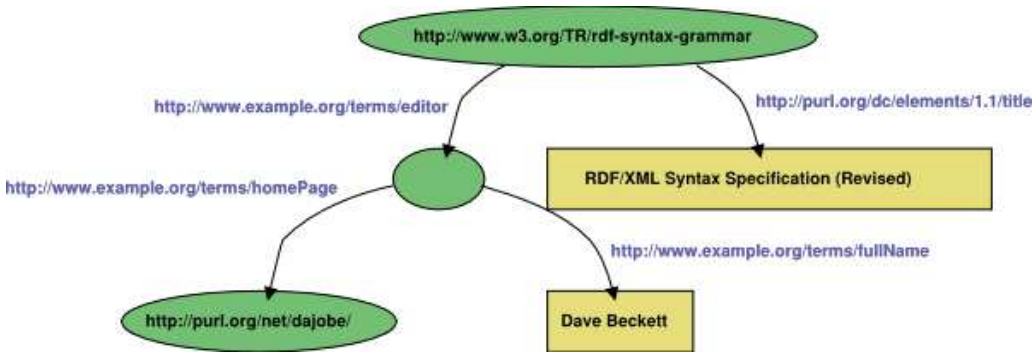

Figure 2.2: RDF graph example.

Web Ontology Language

The first level above RDF required for the Semantic Web is an ontology language that can formally describe the meaning of terminology used in "Web resources", as described above. The Web Ontology Language (OWL) provides an expressive vocabulary for describing properties and classes: among others, relationships between classes, such as disjointness, cardinality, equality, richer typing of properties, characteristics of properties, such as symmetry, and enumerated classes [20].

OWL provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users.

- **OWL Lite** primarily supports classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1.

- **OWL DL** supports the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions. OWL DL is so named due to its

correspondence with Description Logics.

- **OWL Full** exploits the maximum expressiveness and the syntactic freedom of RDF, although it does not provide any computational guarantee.

Let us note that other languages have been proposed and used to define ontologies beyond Semantic Web languages. For example, it is possible to define ontologies by means of well-established logic languages, such as Logic Programming languages as Prolog. However, Semantic Web languages offer the great advantage of providing interoperability and standardization, based on the XML format of serialized RDF or OWL documents. In addition, their increasing success within the research community, both from academia and industry, represents a promising step towards the reach of a shared agreement on semantic interoperable standards.

Context Ontologies

In this section we present some relevant approaches to context modeling that are based on ontologies.

A significant context modeling approach based on ontologies is the CoBrA system [33]. The CoBrA system uses a broker-centric agent architecture to provide runtime support for context-aware systems in ubiquitous computing environments, such as intelligent meeting rooms. CoBrA relies on the SOUPA ontology, which provides a set of ontological concepts to characterize entities such as persons, places or several other kinds of objects within their contexts. SOUPA is developed in OWL.

Another interesting approach has been proposed as the Aspect-Scale-Context Information model [92]. Differently from SOUPA, this model provides its own Context Ontology Language (CoOL), which is supplemented by integration elements such as scheme extensions for Web Services. The CoOL language is used to support context-awareness in

distributed service frameworks for various applications, like for example to check service interoperability in terms of contextual compatibility.

The CONON context modeling approach aims at developing a context model based on ontologies to exploit knowledge sharing, logic inferencing and knowledge reuse capabilities [105]. Similarly to CoBrA, Wang et al. created an upper ontology which captures general features of basic contextual entities and a collection of domain specific ontologies and their features in each subdomain. The CONON ontologies are serialized in OWL-DL. This allows for consistency checking and contextual reasoning using inference engines developed for description logic-based languages.

2.2.3 Context Information Management and Provisioning

Several architectures have been developed to address the issue of collecting, managing and distributing context information to interested applications. Different approaches can be adopted depending on specific application requirements and scenarios, such as the deployment of sensors, the number of potential system users, as well as the technical properties of used devices.

Until now, different categorizations of context management systems have been proposed, none of them being exhaustive nor fully agreed on. The proposed classifications focus on different issues related to context management, such as context acquisition, access and sharing, as well as on architectural properties of the context management support system. In the following we provide an overview of three relevant state-of-art proposals that classify context management systems according to different criteria. It is worth noting that the classification principles characterizing each proposal cannot be considered orthogonal nor clearly distinguished. However, because of the lack of agreement on a common conceptual classification, hereinafter we present them separately.

As far as context acquisition is concerned, Chen presents three possible approaches [33]:

- **Direct sensor access**. This approach is often used in devices with locally built in sensors. The client software gathers the desired information directly from these sensors, which means that there is no additional layer for gaining and processing sensor data. Sensor drivers are hardwired into the application. This tightly coupled approach is not particularly well suited for distributed systems due to its lack of flexibility and reusability.

- **Middleware**. A middleware-based approach introduces a layered architecture in the design of context-aware systems with the intention of hiding low-level sensing details. The middleware is responsible for collecting context information from sensors, storing and aggregating it, and distributing it to interested applications. Compared to direct sensor access, this technique simplifies application development since the client application code does not depend on specific sensors, and it favors the reusability of sensing components that encapsulate sensors.

- **Context server**. The presence of a server permits multiple clients access to possibly remote context data sources. This distributed approach extends the middleware based architecture by introducing a remotely accessible component, the so called *context server*, which gathers sensor data and makes them available to client applications via concurrent, multiple access. Beside the reuse of sensors, the usage of a context server has the advantage of relieving clients from the burden of performing resource-intensive operations. As a drawback, the design of a context-aware system based on client-server requires to consider issues like communication protocols, network performance, quality of service parameters, which characterize a client-server distributed system.

Another interesting classification of context management systems is to be found in [107], where Winograd describes three different context management models for coordinating multiple processes and components. This classification is focused on context access and distribution rather than context acquisition.

- **Widgets**. Derived from the homonymous graphical user interface elements, a widget is a software component that provides a public interface for a hardware sensor [41]. Widgets hide low level details of sensing and ease application development due to their reusability. Widgets are usually controlled by a manager. The tight coupling of widgets with their managers increases efficiency, but leads to a lack of tolerance to component failures.

- **Networked services**. This more flexible approach resembles the context server architecture described above. Instead of a global widget manager, discovery techniques are used to find networked services. This service based approach is not as efficient as a widget architecture due to complex network based components, but provides increased robustness and scalability.

- **Blackboard model**. In contrast to the process-centric view of the widget and the service-oriented model, the blackboard model represents a data-centric view. In this asymmetric approach, applications post messages to a shared media, the so called *blackboard*, and subscribe to it to be notified when some specified event occurs. Advantages of this model are the simplicity of adding new context sources and the easy configuration. Unfavorable is the need of a centralized server to host the blackboard and the lack in communication efficiency as two hops per communication are needed.

Finally, Hong and Landay propose a classification for software systems to support context-aware applications [56]. In this case, by focusing on the architectural properties of the

context management support, they outline four main categories.

- **Libraries**. A library is a generalized set of related algorithms. Libraries are mainly developed to promote code reuse. For example, implementations of the JSR 179 Location APIs for Java 2 Micro Edition include code for manipulating location information within the Java framework [4]. Libraries are generally lightweight and easy to use. However, they tend to be focused on low level context details and do not provide any support for application design.

- **Frameworks**. With respect to libraries, a framework-based approach is more focused on design reuse by providing a basic structure for a certain class of applications, which can be customized according to the application requirements. A relevant example falling into this category is the Java Context Aware Framework (JCAF), a Java-based lightweight infrastructure and programming API, developed by Bardram et al., to support context-aware applications [19]. The aim of JCAF is to let programmers focus on modeling and using context information specific for their application, while relying on a basic infrastructure to handle the actual management and distribution of this information. The main limitation of JCAF lies in the fact that it is bound to a specific programming language and environment, i.e., J2ME, which is not supported by most portable devices, and depends on pre-defined communication protocols that cannot be altered, thus leading to a lack of flexibility in system implementation. In addition, the framework defines its own high level model of context: on one side this might help in the design of a context-aware application, on the other side it prevents the design of possibly needed extensions to the context model itself.

- **Toolkits**. Toolkits are typically built on frameworks and offer a number of reusable components each one addressing a specific functionality. For example, a toolkit might

offer reusable components for accessing sensors and aggregating context information, such as in the case of the well known Context Toolkit from Dey and Abowd, the first comprehensive support toolkit for context-aware applications development [41]. Toolkits represent a significant step towards the realization of reusable context management support systems. However, similarly to frameworks, they typically depend on specific implementation platforms, operating systems and/or programming language. The Context Toolkit already supports some kind of interoperability. The main limitation of a toolkit is therefore its lack of network-based access: implemented as a single application, a toolkit does not provide any support for distributed access and sharing of context information.

- **Infrastructures**. An infrastructure represents a well-established, reliable and accessible set of technologies acting as a foundational basis for other systems. In particular, service infrastructures expose their capabilities as *services*, which can be generally defined as logical units of functionality, usually accessible via a network. Let us note that middleware support solutions are typically implemented as infrastructures. Several infrastructures have been proposed to support context-aware applications. Middleware infrastructures are particularly well suited to support the development of context-aware applications, as we will explain in the following section.

As stated before, the different categories defined in the above classifications might partially overlap. For example, the concept of infrastructure, as described in [56], clearly resembles the networked services defined by Winograd in [107]. In addition, some context acquisition and distribution models are naturally suited to be exploited within some architectural approaches, such as in the case of widgets, which represent the basic components of the Context Toolkit by Dey and Abowd [41].

In general, the choice of an adequate model for context acquisition and management depends on specific application requirements and characteristics, However, we believe that an infrastructure-based approach relying on the underlying network support for distributing and accessing context bring several advantages. First, by providing uniform abstractions and reliable services for common operations, middleware infrastructures facilitate the development of robust applications even on a diverse and changing set of devices and sensors. In addition, they allow sharing and reuse of context information, while carrying the burden of data acquisition, processing and interoperability on behalf of the application. Finally, the modular nature of a middleware infrastructure allows to customize context provisioning and management depending on the specific needs of the client application.

In the following section, we will describe some relevant existing solutions in the area of middleware infrastructures for context provisioning. A more extensive description can be found in [17].

Context Management Middleware Infrastructures

The most common design approach for distributed context management frameworks is a hierarchical infrastructure with several components organized in a layered architecture, as depicted in Figure FIG-1-BALDAUF-LAYERED ARCH. In particular, the *Raw Data Retrieval* layer collects data from sensors on the underlying layer. Let us note that a sensor can be thought of as a programmatic interface that, when queried, returns an answer about a specific context data. This means that, beyond physical sensors, e.g., light and temperature sensors, software applications might also serve as sensors, such as in the case of a user's personal calendar providing information about the user's current activity. The *Preprocessing* layer includes any processing component that takes raw context as an input, performs a data processing activity and provides processed data as an output. Example of

processing activities include context aggregation and verification, as well as inferring additional context information from available data. Once processed, context information is stored and managed by the dedicated layer for further retrieval and access by context-aware applications, which are logically layered on top of the architecture and exploit the underlying components to be provided with needed context information. Let us note that the same logical architecture can be implemented and deployed according to different schemas, e.g., as a centralized server or a distributed system, thus achieving variable levels of efficiency and scalability.

Despite being, strictly speaking, a toolkit, the Context Toolkit represents the first relevant example of infrastructure-oriented support solution for context management [39]. Therefore, its proposed model exerted a prolonged influence on subsequent research in the area of context-aware computing. The system is based on a centralized discovery server where distributed sensor units (called *widgets*), interpreters and aggregators are registered in order to be found by client applications. The toolkit also provides object-oriented APIs to create instances of these components. These components and their respective functionalities set a reference for further research and the layered architecture of Figure 2.3 was developed based on this original model. The SOCAM (Service-Oriented Context-Aware Middleware) project introduced by Gu et al. proposes an architecture for the building and the rapid prototyping of context-aware mobile services [52]. It relies on a central server, called context interpreter, which acquires context data through distributed context providers and provides this information after some kind of processing to interested clients.

Another framework based on a layered architecture is presented in the Hydrogen project, whose context acquisition approach is targeted to mobile devices [55]. The architecture consists of three layers: the adaptor layer, the management layer and the application layer, with analogous functionalities to the ones described in the generic layered

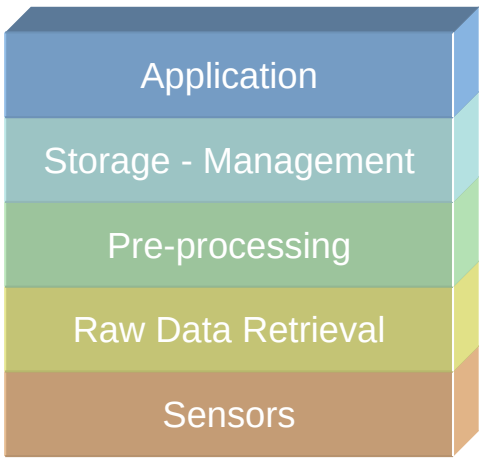

Figure 2.3: Context-aware layered conceptual framework.

architecture (see Figure 2.3). However, differently from most approaches, the Hydrogen system tries to avoid the need to rely on a single centralized server for context acquisition by distributing several context servers on different devices. Devices in physical proximity are in fact enabled to share their contexts in a peer-to-peer manner by exploiting available wireless connectivity options, e.g., 802.11 or Bluetooth. Hydrogen object-oriented context model allows the addition of new context types by specializing the generic context super-class. A notable characteristic of the system lies in the adoption of XML-based formats and protocols for inter-layer communication, thus achieving a certain degree of platform and language independency.

An interesting middleware support solution for distributed context management is Contory [84]. Contory is a middleware specifically designed to accomplish efficient context provisioning on mobile devices. To make context provisioning flexible and adaptive based on dynamic operating conditions, Contory integrates multiple context provisioning strategies, namely internal sensors-based, external infrastructure-based, and distributed provisioning in ad hoc networks. This approach presents two advantages. First, arranging different

context strategies permits to compensate for the temporary unavailability of one mechanism and coping with dynamic resource availability. In addition, combining results collected through different context mechanisms allows the application to partly relieve the uncertainty of a single context source and to more accurately infer higher-level context information. Applications can request context information provided by Contory using a declarative query language, which features on-demand, periodic, and event-based context queries.

In the previous sections we have shown how context information can be represented and managed according to different modeling and design criteria. However, realizing context-aware applications for pervasive environments also requires to design novel middleware solutions that are able to exploit and propagate up to the application level context information, and allow the context-dependent adaptation of applications. The next sections will be therefore devoted to present emerging research guidelines to address the issue of building context-aware middleware that supports application adaptation.

2.3 Metadata-Based Context-Aware Middleware

An emerging approach to support context awareness and to perform application management accordingly is the adoption of *metadata* for representing both context information and the choices in application behavior at a high-level of abstraction, with a clean separation between application management and application logic. Metadata can describe both the structure/meaning of the resources composing a system and the specification of management operations expressed at a high level of abstraction [77].

The effectiveness of the metadata adoption depends on the characteristics of the language used for metadata specification and of the runtime environment for the metadata support. Metadata specification should exploit declarative languages to accommodate users

of different expertise, to simplify metadata reuse and modification, and to facilitate the analysis of potential conflicts and inconsistencies. Metadata runtime support should be responsible for metadata distribution/update and for policy activation/deactivation/enforcement, independently of application logic. The following sections will first provide an overview of emerging metadata specification models, and will then present some relevant examples of existing middleware solutions that exploit metadata to enable the context-aware adaptation of mobile applications.

2.3.1 Metadata Models

Among the different possible types of metadata, profiles and policies are considered of increasing interest and start to be widely exploited in open and dynamic distributed systems. *Profiles* represent characteristics, capabilities, and requirements of users, devices, and service components. For example, markup scheme and ontology-based context representation models described in Section 2.2.1 are typically encoded as profiles. *Policies* express the choices ruling system behavior, in terms of the actions subjects can/must operate upon resources. Profiles and policies are maintained completely separated from system implementation details and are expressed at a high level of abstraction to simplify their specification by system administrators, service managers, and even final users.

Profile Modeling and Representation

Several research efforts are attempting to identify well accepted formats for the most common access devices and spreading standard profile adoption for expressing user needs/requirements. Profile standardization is in fact crucial for resource reusing and sharing in pervasive environments. Most common examples of profiles include *user* and *device* profiles. The former usually describe data about user preferences, interests and demograph-

ics, as well as behavior models. The latter generally contain technical data describing device capabilities, such as available memory, screen resolution and installed software, as well as device status parameters, e.g., battery level. Modeling the context of mobile applications also requires to consider profiling parameters of the *environment*, such as properties of the network connection between the user and the accessed service, and conditions of the user's environment, e.g., light, temperature and weather conditions. In addition, since most mobile applications are designed according to a service-oriented approach, it is also necessary to provide support for *service* profiling. [16]. We do not intend to provide here a comprehensive survey about the state-of-the-art profile modeling solutions, but to outline main research directions and existing standards for profile specification.

Several research efforts and commercial solutions model and exploit user profiles. According to [16], they can be classified by taking into account different dimensions, including the modeling approach to user profiles, the richness and generality of user data included in the model, and the method to acquire data from the user, e.g., by means of explicit user input or by deriving information from user behavior. We do not focus here on the issue of collecting information from the user, instead we refer to the case when information is explicitly gathered from the user, e.g., by direct user input, and exploited to perform some kind of tailoring of applications based on the defined profile. Several systems adopt XML-compliant formats for user profile specification, while others define ontology-based user profiles [34, 14, 22]. For example, the CARE framework, which originally supported only CC/PP-compliant profile data, has been enhanced to allow the specification of ontology-based profiles, encoded in OWL-DL [14]. This allows to choose the most suitable option for profile representation, depending on whether the application needs more expressive power (mainly provided by OWL) or efficiency (favored by the exploitation of a simpler format like CC/PP). Most systems define their own model of user profiles, and a comprehensive

description of existing solutions is out of the scope of this section. Let us note that, although those models tend to be similarly structured, until now no wide agreement on a common standard for user profiling has been reached yet. Some promising solutions are emerging in the field of Semantic Web, which characterize the user in terms of his social relations. For example the Friend-of-a-Friend (FoaF) project describes a user's social network by means of a dedicated RDF ontology [2].

The most prominent solution in device profiling is the Composite Capability/ Preference Profiles (CC/PP) standard, defined by the World Wide Web Consortium (W3C) [67]. CC/PP exploits the XML serialization of RDF (see Section 2.2.2) to allow the creation of profiles describing the capabilities of a device and possibly the preference of its user. CC/PP profiles are structured as sets of components that contain various attributes with associated values. Components and their values are defined in CC/PP vocabularies, specified in RDF, such as the UAProf vocabulary proposed by the Open Mobile Alliance for representing the hardware, software and network capabilities of mobile devices [12]. Let us note that CC/PP, whose first version was recommended as a W3C standard in 2004, is now in the process of being upgraded to benefit from the functionalities of the newer version of RDF. The main advantage offered by CC/PP lies in the great standardization effort that was undertaken by several mobile device manufacturers to reach an agreement on a common format for the representation of device characteristics. However, CC/PP does neither support user nor environment profiling, which hinders its widespread adoption for context and metadata representation.

As far as service profile modeling is concerned, significant efforts have been spent both by academia and industry to define a common representation format for describing services. In particular, the Web services community has been promoting XML-based standard profiles and protocols to describe, search and retrieve services on the Web. The Web

Service Description Language (WSDL), which is now a W3C standard, represents the most significant solution for service profile specification and has been in fact adopted by many companies to allow the development of Enterprise Application Integration solutions (see Chapter 1) [35]. WSDL mainly describes a service in terms of expected input and output, where inputs and outputs are represented by messages, and it also provides a reference, the so-called *grounding*, to the concrete implementation of a service instance. To provide more expressive representation models for services, several semantic-based languages for service description have been proposed, such as OWL-S [44], WSMO [85] and Meteor-S [102], which model both service interface (input/output) and service process workflow by relying on service ontologies.

Policy Modeling and Representation

Policies, which constrain the behavior of system components, are becoming an increasingly popular approach to dynamic adjustability of applications in academia and industry. A policy-based approach to system design and management brings several benefits, including reusability, efficiency, extensibility, context-sensitivity, verifiability, support for both simple and sophisticated components, protection from poorly designed, buggy, or malicious components, and reasoning about component behavior. Policy-based systems generally distinguish two different kinds of policies [90]. *Authorization* policies specify the actions subjects are allowed to perform on resources depending on various types of conditions, e.g., subject identity and resource state; *obligation* policies define the actions subjects must perform on resources when specified conditions occur. Over the last decade policies have been applied to automate network administration tasks, such as configuration, security, recovery, or quality of service (QoS). Multiple approaches for policy specification have been proposed, ranging from interpretable policy languages to rule-based policy notations

(if-then-else), to the representation of policies as entries in a table consisting of multiple attributes [98]. Beyond these traditional applications, new challenges for policy management are emerging. The aim of this section is not to provide a general survey of the state-of-the-art in policy representation, but to describe selected technical aspects of a few policy approaches that have been specifically designed and extensively tested for management of multi-agent and distributed systems in pervasive and mobile scenarios. A more detailed description of relevant policy-based approaches, particularly in the area of security, will be provided in Chapter 5.

Ponder is a declarative, object-oriented language that supports the specification of several types of management policies for distributed object systems and provides structuring techniques for policies to cater for the complexity of policy administration in large enterprise information systems [38]. It has been widely deployed in many applications. A basic Ponder policy is considered a rule governing the choices in system behavior and is specified by a declaration between a set of subjects and a set of targets. These sets are used to define the managed objects that the policy operates over. Ponder uses the term subject to refer to users, principals, or automated manager components, which have management responsibility (i.e., have the authority to initiate a management decision). A subject can operate on target objects (resources or service providers) by invoking methods visible in the target interface. The fundamental policy types in Ponder are obligations and authorizations. Ponder policies also rely on the key concept of management domains. Domains provide a means of grouping objects on which policies apply and can be used to partition the objects in a large system as desired.

To deal with the dynamicity and heterogeneity of pervasive scenarios, novel policy-based systems have emerged that propose a semantic approach to policy definition and management. Semantic technologies permit to represent and reason about policies and

application domains, thus increasing flexibility in policy specification and expressiveness in policy evaluation. A relevant example is the framework KAoS, which provides policy and domain management services for agent and other distributed computing platforms [100, 101]. KAoS has been deployed in a wide variety of multi-agent and distributed computing applications. KAoS policy services allow for the specification, management, conflict resolution and enforcement of policies within agent domains. KAoS is based on an ontological approach to policy specification, which exploits OWL, i.e., description logic (DL), features to describe and specify policies [20]. The KAoS policy ontologies distinguish between authorizations and obligations: a policy constrains the actions that an agent is allowed or obliged to perform in a given context. By relying on DL features, KAoS is able to classify and reason about both domain and policy specification basing on ontological subsumption, and to detect policy conflicts statically, i.e., at policy definition time. However, a pure OWL approach encounters some difficulties with regard to the definition of some kinds of policies-specifically those requiring the definition of variables. For this reason, KAoS developers have introduced role-value maps as OWL extensions and implementing them within the Java Theorem Prover, used by KAoS.

Rei is a policy framework that permits to specify, analyze and reason about declarative policies defined as norms of behavior [60, 59]. Rei adopts a rule-based approach to specify semantic policies. Rei policies restrict domain actions that an entity can/must perform on resources in the environment, allowing policies to be developed as contextually constrained deontic concepts, i.e., right, prohibition, obligation and dispensation. The first version of Rei (Rei 1.0) is defined entirely in first order logic with logical specifications for introducing domain knowledge [60]. The current version of Rei (Rei 2.0) adopts OWL-Lite to specify policies and can reason over any domain knowledge expressed in either RDF or OWL [59]. Though represented in OWL-Lite, Rei still allows the definition of variables

that are used as placeholders as in Prolog. Therefore, Rei's rule-based approach enables the definition of policies that refer to a dynamically determined value, thus providing greater expressiveness and flexibility to policy specification.

2.3.2 Metadata-Based Middleware

To support context awareness and to perform context-aware application adaptation, the emerging research direction is the adoption of metadata for representing both the context characteristics and the choices in service behavior at a high-level of abstraction, with a clean separation between service management and service logic. Section 2.3.1 has provided an overview of emerging models for metadata definition. This section presents significant examples of middleware infrastructures supporting context-aware applications by exploiting metadata. In particular, special attention is devoted to semantic metadata-based middleware.

The GAIA project is a middleware infrastructure that enhances operating system features to include context-awareness [83]. GAIA is built on a CORBA distributed support, but it provides additional features to enable context-awareness. In particular, GAIA defines a common model of context, which all agents can use in dealing with context. This model is based on a predicate model of context and has been supplemented with ontologies to define the semantics of various contexts. It also supports acquisition of contextual information, reasoning about context and modifying agents behavior based on the current context. The model of context and the middleware support the use of different reasoning mechanisms, such as first order logic and temporal logic, to reason about context and decide how to behave in different contexts. Agents can alternatively employ learning mechanisms like Bayesian learning and reinforcement learning to learn different behaviors in different contexts.

The Context Broker Architecture (CoBrA) is an agent-based architecture for supporting context-aware computing in so-called *smart spaces*. Smart spaces are distributed system that consist of communities of intelligent agents, services, devices, and sensors sharing a common goal. For example, in the EasyMeeting application, a smart meeting room is set up to provide relevant services and information to the meeting participants (e.g., speakers, audiences, and organizers) based on their situational conditions (or contexts) [34]. CoBrA uses Semantic Web languages for representing context ontologies and for supporting context reasoning, thus facilitating independently developed agents to share context knowledge and minimizing the cost of context sensing. Central to CoBrA architecture is the presence of a Context Broker, an intelligent agent that runs on a resource-rich stationary computer in the space and gathers raw context data from sensors deployed in the environment. The context broker is also responsible for reasoning about the information that cannot be directly acquired from sensors, detecting and resolving inconsistent knowledge that is stored in the shared model of context, and protecting user privacy by enforcing policies (specified in the Rei language [60]).

Agostini et al. proposed the Context Aggregation and REasoning (CARE) middleware to support a set of requirements for context-awareness in distributed environments. CARE has three major goals, namely: supporting the fusion and reconciliation of context data obtained from distributed sources, supporting context dynamics through an efficient form of reasoning, and capturing complex context data that go beyond simple attribute-value pairs. CARE adopts profiles and policies to perform context-based tailoring of service provision to mobile users. In particular, as described in Section 2.3.1, profiles are represented by using Composite Capability/Preference Profiles (CC/PP) [67]. Profiles include both shallow context data and ontology-based context data which are expressed by means of references to ontological classes and relations inserted in the CC/PP profile. The choice

of defining two different kinds of profiles is motivated by performance issues reported by the authors in [15]. Each entity has a dedicated Profile Manager for handling its own context data. Both the user and the service provider can declare policies in the form of rules over profile data which guide the adaptation and final personalization of the service. The Context Provider module is in charge of building the aggregated context information for the application logic. In particular, it evaluates adaptation policies and solves possible conflicts arising among context data and/or policies provided by different entities.

CARMEN is a middleware for context-aware resource management that supports and facilitates the design, development, and deployment of context-dependent services for the wireless Internet [22]. CARMEN allows service providers, system administrators, and final users to specify different kinds of metadata in a declarative way at a high level of abstraction. CARMEN metadata influence the dynamic determination of context and, consequently, the context-based service provisioning, without any intervention on the application logic, according to the design principle of separation of concerns. CARMEN exploits two types of metadata: profiles to describe the characteristics of any resource modeled in the system, and policies to manage migration, binding and access control. CARMEN adopts XML-based standard formats for profile representation to deal with the Internet openness and heterogeneity, such as CC/PP for user/device profiles [67] and WSDL for the service component interface description [35]. In addition to profiles, CARMEN expresses policies as high-level declarative directives: access control policies to ensure secure resource usage and mobility handling policies to guide the middleware decisions in response to context variations at service provision time. Policies are specified in the Ponder policy language. The CARMEN middleware is designed according to a layered architecture, where the higher level layer is responsible for managing metadata and context, and the lower level layer is in charge of providing common features for distributed service provisioning, e.g., support for

naming, event management and distributed communication. A peculiar feature of CARMEN is the way it handles dynamic variations in service provisioning due to user mobility, i.e, by exploiting mobile agents, called shadow proxies, that migrate across nodes on the fixed network and act on behalf of mobile users.

2.4 Alternative Design Guidelines for Context-Aware Middleware

Section 2.3 has focused on the emerging guideline of adopting metadata to specify and enforce behavior adaptation in response to context variations. Alternative approaches have been proposed to address the same issue, being the following the most significant approaches: reflective middleware and middleware based on dynamic aspect-oriented programming paradigm.

2.4.1 Reflective Middleware

To support adaptation features, several research efforts have proposed the adoption of reflection techniques in mobile middleware design and development. *Reflection* is a design and programming technique providing principled mechanisms to inspect the structure and behavior of a system and make changes to both at run time [48]. For this purpose, a reflective system maintains a representation of itself that is casually connected to the underlying system it describes, the so called *casually connected self-representation* (CCSR) (citare 6 e 7 di [48]). CCSR is often referred to as the *meta level*, while the system itself is the *base level*. Hence, any change made at the meta level via this self-representation are reflected in the underlying base level, and vice versa. The process of making the base level tangible and accessible at the meta level is known as *reification*, while operations to introspect and make

changes to the meta level are commonly referred to as the *Meta Object Protocol* (MOP).

Reflection has been predominantly applied to language design, thus leading to the definition/extension of many languages, such as Sun's Core Java Reflection library (citare 8 del capitolo) and OpenC++ (citare 10), and later to the field of operating systems (citare 78 di [72]. In this section, we focus on the use of reflection in middleware design.

Reflection has been applied to build traditional middleware, such as in the case of OpenORB (22 di [72]), OpenCORBA (36 di [72]), and dynamicTAO (34 di [72]). More recent approaches have investigated the use of reflection in the design of mobile middleware solutions to achieve context-awareness and adaptation mechanisms [86]. In those systems, reflection is used to make both the internal structure of the middleware and its behavior visible, and to adapt the middleware behavior to changes in the execution context, e.g., available resources in the locality where a mobile device is currently attached and interaction protocols needed to cooperate with other middleware components. The latter is particularly relevant to address interoperability issues arising from heterogeneous middleware platforms that need to interact, such as in the scenarios targeted by the ReMMoC framework [49]. ReMMoC is a reflective middleware framework that supports the capability to develop applications independently of middleware implementation. The framework combines the Lancaster approach of components, component frameworks and reflection with a Web Services based abstraction. The CARISMA framework, developed at University College London, is a peer-to-peer middleware for context-aware service provisioning [32]. CARISMA adopts a mixed approach, based both on reflective techniques and metadata. Nodes can select services according to user and application context information, which is represented by means of profiles and policies. In particular, different policies define how to provide a specific service based on different context conditions, like for instance the variable amount of network bandwidth. Reflection is used in CARISMA to inspect and adapt poli-

cies to the current context: for example, depending on available bandwidth, the middleware selects and enforces the most appropriate policy to allow the transfer of a file. Another middleware for mobile environments that relies on a combination of reflection and metadata is ALICE, developed at Trinity College Dublin [26]. ALICE supports network connectivity in mobile environments by providing a range of client-server protocols and selecting the most appropriate one to current context, i.e., network connectivity status, according to configuration policies specified in the Chisel policy language.

Reflection represents an interesting design guideline to achieve context-awareness in middleware solutions since it provides powerful dynamic adaptation features. However, reflective middleware provides little support to adaptation control and management. This is particularly important with respect to context-aware middleware since it is crucial to clearly define, retrieve and classify adaptation strategies based on context variations. In addition, the integration of reflective middleware with legacy systems that are typically implemented in non-reflective programming languages poses significant limitations to the widespread adoption of this middleware design approach.

Let us note that context metadata can be viewed as a form of structural reflection, providing additional (meta) information about the underlying system, e.g., physical location, current battery levels, network bandwidth and performance) [48].

2.4.2 Aspect-Oriented Middleware

Aspect-oriented programming (AOP) is a software engineering approach designed to support the implementation of *cross-cutting* concerns, i.e., system properties and functions that cannot cleanly declared in terms of systems components since they must be scattered or distributed across otherwise unrelated components [63]. Typical examples include features such as security, persistence, logging and monitoring. In other design approaches

developers often implement these features in an ad-hoc manner across the system, thus leading to increased system development, debugging and evaluation time. AOP supports the concept of separation of concerns to counter this problem: instead of implementing cross-cutting features within the base code, AOP models them as *aspects*, i.e., pieces of code that can be woven into the base code at compile time. Dynamic AOP programming provides even greater benefit since aspects can be woven into base code at run time.

Dynamic AOP techniques have been exploited in the design of mobile middleware to achieve dynamic behavior adaptation [61]. For example, the MIDAS middleware, built on top of the dynamic AOP-based framework PROSE, provides runtime extensions to mobile applications that might need additional functionalities to adapt to different environment conditions, such as encryption layers or billing modules [82]. Lasagne is an AOP framework that supports the dynamic customization of middleware platforms and distributed systems [99]. The aspect-oriented approach of Lasagne is based on *extensions*, i.e., code units that can be dynamically introduced in the system to update the behavior of multiple components. The interesting feature of this framework is that extensions are selected based on context information, where contextual properties are defined and attached to specific service functionalities. This allows some kind of context-aware adaptation of the system's behavior.

Let us note that the AOP and the reflection-based approaches represent possible solutions to the issue of adapting the middleware behavior by taking different perspectives. AOP is basically a set of software engineering techniques, which allows the modeling of the middleware structure at a high level of abstraction, based on the assumption that the engineering of some "aspects" of a system cannot be hard-coded into the application logic at design time. On the contrary, reflection is a programming principle that enhances software objects with the ability to inspect their own qualifying properties. Therefore, to a certain

extent, these two approaches might be considered complementary since the former pertains to the engineering process and the latter to the programming phase.

Due to the ability of supporting dynamic behavior adaptation, dynamic AOP appears to be a promising area of research for the design of context-aware mobile middleware. Analogously to reflection, however, AOP methodologies lack a structured mechanism to both specify the exact code location where adaptation is needed, and how adaptation should be applied depending on user, application and environment context. Metadata-based middleware solutions address this issue by specifying adaptation strategies using declarative metadata.

2.5 Chapter Summary

In this chapter we have provided an overview of existing technologies and solutions to (i) represent context information, (ii) collect, manage and distribute context information to interested applications, and (iii) support adaptation of pervasive applications in response to context changes. In particular, with regard to context representation, we have shown the increased expressivity and flexibility offered by an ontology-based representation of context, with special attention to Semantic Web standards. We have also explained the advantages in terms of reusability and robustness brought by the adoption of a middleware infrastructures for context management. Finally, we have discussed the emerging research direction of adopting metadata in middleware design, which supports context-awareness and application adaptation, while keeping a clean separation between application logic and management.

Chapter 3

Towards Semantic-Enabled Context-Aware Middleware

The purpose of a computer is to help you do something else.
Mark Weiser

The previous chapter has provided an overview of different approaches to the design of context-aware middleware. This chapter will illustrate the main guidelines we have followed in this thesis to design and develop novel context-aware middleware support solutions. In particular, it will first show the limitations of existing solutions, thus providing motivations for our choice of adopting semantic metadata to support context-awareness. Then, it will present some relevant pervasive application areas, which would benefit from being enhanced with support for context-awareness. These specific application areas have been selected because they represent crucial steps towards our envisioned pervasive scenarios, where mobile users dynamically interact with each other, by exploiting available resources via possibly heterogenous devices and under variable conditions. For each described application area, we

have designed and implemented a semantic-enabled context-aware middleware architecture, as described in the next chapters.

3.1 Enhancing Mobile Middleware with Explicit Semantics

Pervasive applications are characterized by a dynamic and constantly changing execution context. This raises new challenges for the design of mobile middleware solutions, which should support mobile computing applications in the task of adapting their behavior to frequent changes in the execution context, i.e., middleware should become context-aware. Recalling Chapter 1, designing novel middleware solutions for pervasive environments is a demanding task as it requires to address some crucial requirements. In particular:

- *Middleware solutions should provide expressive representation models that allow to describe interacting entities, their operating conditions and the surrounding world, i.e., their context, according to an unambiguous semantics.* This allows interoperability between possibly unknown entities wishing to dynamically establish an interaction, although originally designed as parts of independently developed applications.

- *Middleware solutions should support distributed applications in the task of reconfiguring and adapting their behavior/results to ongoing context changes, by providing the application with context-awareness.* Context-aware adaptation strategies should be expressed at a high level of abstraction by cleanly separating application management from application logic.

As shown in the previous chapter, the adoption of metadata represents an emerging approach to the design of context-aware middleware solutions. Metadata allow the representation of both context information and choices in application behavior at a high-level of abstraction, with a clean separation between application management and application logic.

This separation of concerns is crucial to reduce the complexity of developing applications for pervasive environments and to favor rapid application prototyping, runtime configuration, and maintenance. Metadata-based middleware solutions thus represent a promising option to address the requirement of supporting context-aware applications.

The effectiveness of the metadata, however, largely depends on the characteristics of chosen languages and tools for metadata specification and runtime exploitation. In particular, a crucial issue to achieve of context-awareness is the ability of the middleware support to properly describe and interpret context information, such as the entities that characterize the system, the interactions occurring between them and the operating conditions under which such interactions occur, as well as context-driven adaptation directives. In addition, given the intrinsic openness of pervasive application scenarios, dynamic interactions between entities sharing little or no prior knowledge about each other are extremely likely to happen. This requires to unambiguously define the meaning, i.e. the *semantics*, of used metadata: based on metadata information, independently developed applications should be enabled by the middleware platform to dynamically interoperate with minimal human intervention.

In most current middleware solutions, however, the meaning of used metadata is only known to developers and/or system administrators: metadata are represented and encoded in a form that is primarily intended for human comprehension and exploitation. The shared assumption is that the conceptual model underlying context description and management is essentially *implicit*, i.e., it is only known to humans who develop the middleware platform, and possibly encoded in natural language, which is clearly not machine-understandable.

We believe that the inability of the middleware platform to automatically acquire and process knowledge about the underlying system has hindered the full achievement of

context-awareness in pervasive applications. The first limitation regards the possibility to automatically process metadata, including automated reasoning, to evaluate and make decisions based on current context information. Automated reasoning over metadata defined with unambiguous semantics might support the middleware in the task of managing system entities according to the context information carried by metadata, while facilitating the analysis of potential conflicts and inconsistencies. The second main limitation of current metadata-based solutions regards cross-interoperability between different middleware platforms. As long as metadata are represented according to models and formats that are only understandable by human users, middleware platforms will not be able to exchange context information outside their own boundaries. Therefore, we claim that novel middleware should support context-awareness by exploiting *semantic metadata*, i.e., metadata whose meaning is explicitly defined in a machine-understandable form and can thus be acquired and processed by software applications.

Following the above considerations, we have derived two main guidelines for the design and development of middleware supporting pervasive applications, namely:

1. Support for **context-awareness**. Novel mobile middleware solutions should support context information representation and management, and propagate context visibility up to the application level, while providing expressive and flexible means to specify context-dependent adaptation strategies.

2. Adoption of **semantic metadata**, that is, metadata whose meaning is unambiguously defined in a machine-processable form. Middleware should be enabled to reason about metadata describing system entities and their context, to take appropriate management decisions based on changing context conditions.

These guidelines have been applied to different application areas within pervasive computing that would particularly benefit from the exploitation of context. Common to all applications is the key role of context in enabling mobile users to personalize applications based on their needs and current situation. In particular, our envisioned scenario is a pervasive environment, where mobile users dynamically interact with each other, by exploiting available resources via possibly heterogenous devices and under variable conditions. To enable this kind of scenario, middleware support should be provided to address some crucial issues, namely:

1. **Service Discovery.** Mobile users should be enabled to dynamically search and retrieve resources/services they need to accomplish their goals and activities. In a pervasive environment, several services with different characteristics are offered to mobile users via different connectivity options and heterogeneous devices. Mobile users should be supported in the task of looking for only those services that are of potential interest based on current context information, like for instance user interests and preferences, device technical properties and service status.

2. **Access Control.** After retrieving needed services and resources, mobile users should be supported in the task of properly access them. Let us note that in pervasive environments mobile users can act not only as service clients, but also as service providers, by making resources and/or functionalities hosted on board of their devices available to other users. Given the high dynamicity of pervasive environments, adequate support should be provided to ensure control over service and resource access. Proper access control permits the secure interaction of mobile users wishing to dynamically interact by reciprocally sharing and exchanging resources.

3. **Social Computing.** Once empowered to securely discover and manage access to

each other's resources, mobile users can exploit the full potential of pervasive application scenarios. For example, by establishing impromptu collaborations users can dynamically exchange knowledge and interact to achieve common objectives. It is therefore necessary to provide support for social computing applications, such as the creation and management of social networks based on user location, interests and social activities, i.e., user context.

We have designed and implemented a semantic-enabled context-aware middleware architecture targeted at each of the above described application area. In the following, we show how our design guidelines, as defined above, can address the specific issues of each application class.

3.2 Personalizing Discovery of Pervasive Services

In the emerging pervasive scenarios, mobile users are able to access several services in any way at any time anywhere, by exploiting all connectivity capacities provided by their portable devices. Therefore, to support mobile user activities, it is crucial to enable the dynamic retrieval of available services in the neighborhood of the user current point of attachment, while minimizing user involvement in service selection, configuration and binding. Service discovery in pervasive environments, however, is a complex task as it requires to face several technical challenges at the state of the art, such as user/device mobility, variations (possibly unpredictable) in service availability and environment conditions, and terminal heterogeneity. Users might need to discover services whose names and specific implementation attributes cannot be known in advance, while service providers need to advertise services to clients whose technical capabilities and conditions at interaction time might be mostly unpredictable beforehand. In addition, service providers cannot exactly

define and code at design time all possible configurations of devices accessing the service, e.g., by including any possible discovery protocol and data format.

Most common discovery solutions have been designed for traditional distributed enterprise/home environments and mobile computing scenarios. The shared underlying assumption is that services can operate in dynamic heterogeneous environments with varying users, devices, service components, and network conditions, but within well-defined boundaries under the management of system administrators. In particular, all existing solutions implicitly define the boundaries of service discovery searching space (discovery scope), with different approaches. One class of solutions, such as Intentional Naming System (INS) and Bonjour, considers administrative domains as the implicit discovery scope: clients can search only within the collection of services under the control and responsibility of their same administrator. Another set of approaches adopts a network topology-based approach to fix search boundaries, such as DEAPspace and BluetoothSDP that include services within a single-hop wireless network range. Other discovery proposals, such as NinjaSDS, SLP, and Jini, slightly extend the previous approaches with query services that allow users to specify either their roles or physical locations to refine the discovery scope. However, different nearby users have the same views of available services regardless other high-level attributes, such as their specific preferences, temporal conditions, and on-going activities.

Ubiquitous computing environments are far more dynamic and heterogeneous than traditional deployment scenarios, thus posing unique discovery challenges. Services can be neither tailored in advance to answer all user needs nor statically configured to fit the characteristics of all access devices. In addition, it is inappropriate to define discovery scopes only on the basis of network topology or administrative domain criteria. Services operate on multiple coexisting networks, which might be only temporary connected, and in different administrative domains. It is also difficult to assume standardized naming

and attribute representations: services are typically administered by different entities, each autonomously assigning its descriptions of service properties. For instance, let us imagine a catering service and a boat maintenance one: while they probably share the concepts of price and availability, they are likely to differently represent them, thus making difficult traditional request/offer matching based on syntactic attribute comparison.

We claim that the specific characteristics of ubiquitous environments should affect service discovery, especially discovery scope and retrieval. Primarily, there is the need for a paradigm shift from network/administrative *domain-centric* to user *context-centric* discovery. Network topology or administrative domain parameters are too coarse-grained to properly define discovery scope boundaries and to automatically select retrieval results in ubiquitous scenarios. The user and her context, as defined above, should be considered central. Novel discovery solutions should fully exploit user context awareness, in an effective and efficient way, to properly determine discovery results. This could also help clients in saving time and efforts in discovered service selection: searching scopes should automatically be limited to the only relevant targets depending on user context.

In addition, because of the impossibility to make a-priori assumptions about the way user context and services are described in an open and dynamic deployment scenario, the other emerging requirement is the adoption of semantic languages. The primary advantage is that semantic technologies permit a formal representation of user context and service properties at a high level of abstraction. On the one hand, that enables automated reasoning on context/service representations. On the other hand, it facilitates interoperability between entities that may wish to interact even if statically unknown. In particular, service retrieval can primarily benefit from a semantic-based approach: traditional discovery queries based on simple naming templates are likely to fail in ubiquitous environments because users typically cannot have total/partial knowledge about needed service identifiers.

Moreover, discovery matchmaking approaches based on service attributes are insufficient because they rely on the exact matching of patterns/keywords.

The lack of existing middleware discovery solutions addressing the aforementioned issues have motivated us to design and implement a novel discovery framework, called Middleware for Intelligent Discovery of context-Aware Services (**MIDAS**). MIDAS supports user-centric discovery by providing mobile users with service views, i.e., set of accessible services, that are personalized based on users' current context. To achieve such context-awareness, MIDAS relies on a semantic metadata representing the properties of interacting entities, and it exploits automated reasoning to match user requests against service offers. A detailed description of the MIDAS architecture is provided in Chapter 4.

3.3 Controlling Access to Resources in Spontaneous Collaborations

Pervasive computing scenarios enable mobile users in physical proximity of each other to form ad-hoc networks for spontaneous collaborations and to engage in opportunistic and temporary resource sharing without relying on the availability of a fixed network infrastructure.

However, ad-hoc collaborations raise new security challenges with regard to the retrieval and use of distributed resources, undermining several assumptions of traditional access control solutions. Traditional access control solutions usually assign permissions to principals depending on their identity/role. In the new pervasive scenario, however, users typically share services with unknown entities and, more importantly, with entities whose identity may not be sufficiently trustworthy. In addition, since spontaneous collaborations are typically established in an impromptu and opportunistic fashion, it may not be possible

to rely on formal collaboration agreements to decide who can access which resources and how, thus excluding the possibility to exploit access control policies defined on a contractual basis as in medium or long-term inter-organizational coalitions. Access control in spontaneous coalitions is further complicated by the high dynamicity in resource availability. Each collaborating entity may alternatively play the role of either a service client or provider or both, depending on dynamic conditions and the current status of interaction. When playing the service provider role, an entity may introduce new services into the environment, thus changing the set of available resources. Variations in resource availability occur also because of the transience of ad-hoc coalitions where entities-resource providers- leave and/or enter a coalition, unpredictably, at any time.

Appropriate access control models are needed to enable resource sharing and access in spontaneous coalition scenarios. It is crucial that the definition and enforcement of access control policies take into account the heterogeneity and dynamicity of the environment in terms of available services, computing devices, and user characteristics.

To address these issues, we advocate a paradigm shift from *subject-centric* access control models to *context-centric* ones. Differently from subject-centric solutions where context is an optional element of policy definition that is simply used to restrict the applicability scope of the permissions assigned to the subject, in context-centric solutions, context is the first-class principle that explicitly guides both policy specification and enforcement process and it is not possible to define a policy without the explicit specification of the context that makes that policy valid. Instead of assigning permissions directly to the subjects and defining the contexts in which these permissions should be considered valid and applicable, a system administrator defines for each resource the contextual conditions that enable one to operate on it. When an entity operates in a specific context, she automatically acquires the ability to perform the set of actions permitted in the current context.

In addition, we consider context crucial for enabling policy adaptation. In pervasive environments the conditions that characterize interactions between users and resources may be largely unpredictable. Consequently, policies cannot all be specified a priori to face any operative run-time situations, but may require dynamic adjustments to be able to control access to resources. We use the term "policy adaptation" to describe the ability of the policy-based management system to adjust policy specifications and evaluation mechanisms in order to enable their enforcement in different, possibly unforeseen situations. In this scope, it is crucial to be able to represent the various operative conditions under which policies should be applied, i.e., the context, and to define the expected behavior of the policy framework on the basis of such context variations.

We also claim that context-centric access control solutions need to adopt ontological technologies as key building blocks for supporting expressive policy modeling and reasoning. Semantically-rich policy representations permit description of policies at different levels of abstraction and support reasoning about both the structure and properties of the elements that constitute a pervasive system, i.e., the context and the management policies, thus enabling policy analysis, conflict detection, and harmonization.

We have implemented these guidelines in the **Proteus**[1] policy framework that exploits (i) context-awareness and (ii) semantic technologies for the specification and the evaluation of access control policies. The Proteus access control framework and its prototype implementation are presented and discussed in Chapter 5.

[1]Proteus is the name of a marine god of the ancient Greek mythology that was able to change his shape into different forms.

3.4 Building Anywhere and Anytime Social Networks

Sociality characterizes an individual's life: people go to bars and restaurants, study together in schools, participate in art, music, and sport groups, form clubs and associations, and work together in teams on production lines and in business. Everybody takes part and plays a role within the framework of a social community. Social ties, such as friendship, similarity of interests or professional activities, compose a web of social binds among individuals which is referred to as *social network* (SN) [36]. Since from early 1960s several research efforts have been spent around the investigation of advanced collaborative systems leveraging human's connections and sociality. Internet-based SN computing solutions allow to establish and maintain on-line virtual social communities of users grouped by common relations, such as common music preferences or job connections. The underlying assumption is that social relations can be established independently from physical places, and the physical places and the social spaces where interactions occur can be maintained separate.

Technology advances in wireless networks and the increasing diffusion of portable/ wearable devices with both fixed and wireless connectivity offer a unique chance to further improve social networking services and to extend their scope of applicability. The possibility of ubiquitous computing of being connected anytime and anywhere enables serendipitous social encounters between proximate users with common interests and the formation of ad-hoc spontaneous SNs, anywhere and anytime [81, 46]. For example, users could exploit their wireless-enabled portable devices to be informed about the availability of their friends in the nearby. Users could also rely on their devices to gather information about people they regularly observe and encounter while on the move, e.g., people who catch the same train every morning, but who do not know yet, i.e., familiar strangers [58].

Ubiquitous computing technologies allow the design/development of novel social

networking services that permit users both to reinforce their existing social binds and to establish new face-to-face ones. In particular, ubiquitous technologies promote a focus shift from virtual to physical social spaces, re-establishing the connection of SNs to physical spaces. This is a key opportunity considering that user proximity and physical places affect and influence social behavior in many ways [58]. Physical proximity, in fact, increases the likelihood of forming impromptu social relationships. In addition, physical places can act as social filters for people; places like museums, discos, centered around specific activities, group together users who are likely to share common characteristics and preferences. Several prototypes of social systems have recently emerged that exploit not only social preferences, but also co-location and/or reciprocal proximity of individuals as key design principle for guiding SN formation/management strategies and for restricting the scope of user interactions [58, 36].

However, to realize the full potential of anywhere and anytime social computing various technical challenges must be still addressed. Anytime and anywhere social computing requires several support mechanisms and tools, including location/proximity systems that permit to identify where users are located and who are nearby, expressive representation models of physical place and user characteristics, support facilities for the retrieval of the information characterizing places/users, and effective social matching algorithms that exploit the visibility of user location/proximity and of place/user characteristics to extract meaningful SNs. Moreover, because of the impromptu and transient nature of ubiquitous interactions, another main challenge is to develop solutions able to extract SNs autonomously and transparently from users by minimizing user intervention. Achieving anytime and anywhere SN computing requires also shared and interoperable vocabularies for modeling location/entity characteristics to avoid inconsistent interpretations typically arising in open and heterogeneous ubiquitous environments.

Current state-of-the-art solutions tend to address only a subset of the above-described social networking management aspects. In addition, the mostly adopted approach in currently available social networking solutions is to embed into the application the handling of location tracking, of relation extraction, and of visibility of SN members. This significantly increases the complexity and the costs of designing, developing and deploying anytime and anywhere social networking services, thus slowing down their widespread acceptance and diffusion.

We claim that the success of anytime and anywhere social computing depends on the design and development of middleware-level solutions integrating within the same framework all the basic facilities needed to support SN management. Anytime and anywhere social services in different application deployment domains would all benefit from similar sets of basic support functions. Middleware-level solutions promote clear separation between SN management concerns and application requirements, thus simplifying social computing application prototyping.

We also believe that context-awareness should guide the design of novel middleware-level solutions for social computing. To support the creation of SNs that reflect the reality of social interactions in ubiquitous environments, it is essential to take into account several context information, such as user location and/or reciprocal proximity, user attributes, motivations, attitudes, activities and social preferences. Toward this goal, middleware proposals should provide integrated support for context modeling, acquisition, reasoning and for context-aware SN extraction.

In particular, it is crucial to find adequate expressive means for rich and unambiguous representation of users, their contexts and the networks they participate in [93]. However, the impossibility to make a-priori assumptions about the way user contexts are described in an open and dynamic deployment scenario, such as the ubiquitous one, compli-

cates context modeling endeavors. Semantic languages seem to offer a promising solution to the key issue of describing social contexts at the proper level of abstraction, while enabling automated reasoning on context representations. In addition, emerging ontology standards, such as RDF and OWL, allow interoperability between possibly unknown users that may wish to establish a social interaction.

Based on the above guidelines, we have designed and implemented a middleware-level solution, called Socially-Aware and MObile Architecture (**SAMOA**) for anytime and anywhere social application provisioning that supports the creation and management of social networks by taking into account users' context, e.g., user preferences, location and execution environments. A key feature of SAMOA is the adoption of a semantic-based modeling approach to context information and a semantic-based social matching algorithm to infer relations among co-located individuals. The SAMOA architecture and its prototype implementation are described in Chapter 6.

3.5 Chapter Summary

In this chapter we have summarized the main guidelines that we have adopted in this thesis to design and develop novel middleware solutions for supporting applications in pervasive environments. In particular, these guidelines are the enhancement of mobile middleware with support for context-awareness and the adoption of semantic technologies to both represent and reason about context. We have also shown how to exploit these ideas in different research areas to provide users with a personalized experience of pervasive applications. In particular, we have implemented a set of middleware-level solutions to (i) support mobile users in discovering services of interest, (ii) securely access those services and resources, and (iii) build anytime anywhere social networks to allow dynamic collaboration

between mobile users based on their current context.

Chapter 4

The MIDAS Service Discovery Framework

Calm technology increases our knowledge and our ability to act, without increasing information overload.
Mark Weiser

This chapter presents our middleware-based approach to support semantic discovery of context-aware services, called Middleware for Intelligent Discovery of context-Aware Services (MIDAS). After a brief introduction to a real application scenario that motivated our work, the chapter describes MIDAS metadata model and middleware architecture. Then, it describes a prototype implementation of the system, discusses its usability through a case study and provides experimental results to show the feasibility of our approach. The chapter ends with an overview of relevant related work and providing some insights into ongoing work.

4.1 Motivating Scenario

To introduce some service retrieval/access issues for next generation mobile systems, let us start by considering the ubiquitous provisioning scenario of a harbor. During high season, a harbor in a crowded vacation locality, such as one in the Tyrrhenian sea, hosts hundreds of boats, leaving and entering the port, with tourists on board who may be willing to access services offered by the harbor computing infrastructure from ubiquitous attachment points (devices embedded into boat equipments, cellular phones, palmtops). For instance, while approaching Capri port, Bob might need to access a booking service for mooring or a route assistant service to safely drive his boat into its assigned place. Once docked, he might benefit from info-stations with the list of all available marine facilities or from tourist agency services to discover local tourist attractions. In addition, during navigation, Bob's boat should have the possibility to interact with other crossing-by ships serving as both service clients and providers. For instance, while sailing toward Ischia, Bob might cross a yacht along its route and desire to exchange tourist information about the already visited marines (photos, suggestions for good restaurants and not crowded bays, people from the same country met in other harbors). It is crucial that users can seamlessly discover all and only the available services of potential interest, by minimizing user involvement in system configuration and service selection.

Service discovery is a complex task in dynamic heterogeneous environments and requires facing several technical challenges at the state of the art: mobility, service availability changes, user role variations, environment unpredictability, and terminal heterogeneity.

Boats move and frequently change their physical position, both with respect to their piers and to other boats in the nearby, which could play the role of service clients/ providers. That implies that devices and services, either deployed over the craft or used

by people on board, often experience variations in service availability due to disconnections and reconnections to possibly heterogeneous wireless networks. New services may also be frequently added in the harbor info-stations, especially in high season, as well as existing ones may be removed, modified, or relocated.

In addition, it may happen that the same subject plays the role of either service user or provider or both: roles are not always statically assigned and might change depending on the current status of interaction. Service discovery is further complicated by the impossibility to foresee not only all the possible interactions that may take place among users and services, but also the environment situations where interactions might happen. Let us consider, for instance, a service that provides incoming tourists with maps of local attractions and instructions to reach them. The service provider cannot exactly define and code at design time all possible configurations of devices that are going to access the service, by including any supported discovery protocol and any possible format for the provided geographic indications (map images, plain text, speech-based,).

Users need to discover services whose names and specific implementation attributes cannot be known in advance, and providers need to advertise services to clients whose technical capabilities and interaction situations are mostly unpredict-able beforehand. Finally, tourist characteristics, requirements, and preferences may be significantly heterogeneous, as they might speak different languages and exhibit a variable degree of technical expertise. Users may also access services in different ways, on the ground while walking around the harbor, or from boats while engaged in other activities, e.g., while sailing or driving the motorboat within the port area. This means that discovery often occurs in situations where users cannot pay much attention to careful and long operations of service retrieval and selection. In addition, access terminals usually exhibit relevant differences in resource capabilities (display size and resolution, computing power, memory, network bandwidth, battery). For

instance, a yacht might be equipped with a global positioning system em-bedded within an onboard personal computer, while a tourist traveling on a rented motorboat can rely only on her limited smart phone capabilities.

4.2 Overview

To address the above issues, we have designed and implemented a context-aware semantic discovery framework, called **M**iddleware for **I**ntelligent **D**iscovery of context-**A**ware **S**ervices (**MIDAS**)[1].

As a key feature, MIDAS uses context awareness to personalize service discovery along two different directions. On the one hand, upon explicit user requests, the middleware exploits user contexts to determine the applicable discovery scopes, thus creating personalized lists of available services (service views, see Figure 4.1). On the other hand, once determined initial service views, MIDAS exploits context awareness to autonomously update views in an effective way, triggered by the only context changes of interest for the currently supported views. View adaptation does not require any user involvement, thus providing the additional value of updated perception of available services without any user overhead [23].

To support context-aware semantic discovery, the two original and key aspects of MIDAS are its full exploitation of semantic metadata and its context-aware facilities.

In designing our middleware, we have also considered the issue of making semantic-based discovery facilities accessible to resource-constrained devices. To this purpose, we have designed an extension of the MIDAS middleware architecture, called **A**daptable **I**ntelligent **D**iscovery of context-**A**ware **S**ervices (**AIDAS**), which integrates MIDAS discovery features

[1] http://www.lia.deis.unibo.it/research/MIDAS/

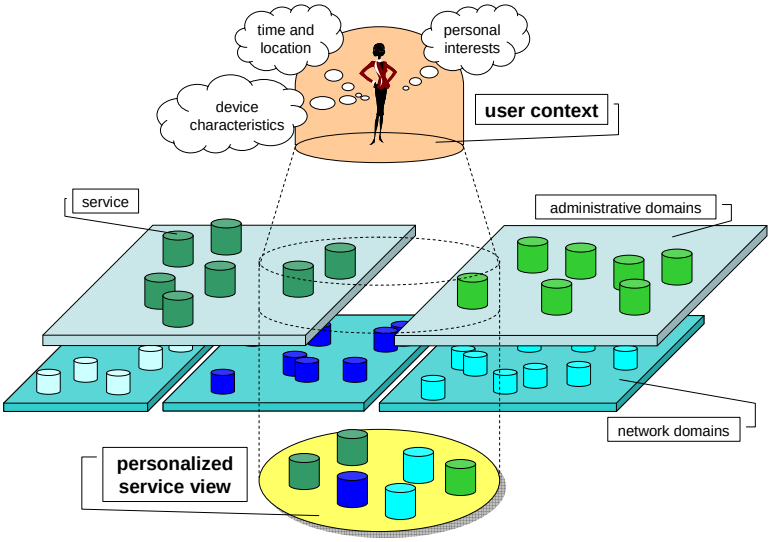

Figure 4.1: MIDAS user-centric service view based on user context and semantic metadata.

with support for the dynamic configuration of semantic facilities to fit the technical capabilities of heterogeneous devices. In particular, AIDAS offers a wide set of mechanisms middleware facilities capable of adapting semantic support to the different characteristics of mobile devices and of providing mobile devices with visibility on semantic functionalities hosted by nearby devices. Hereinafter we describe the MIDAS framework, while providing some insights about the AIDAS extension only in Sections 4.4 and 4.5.2. More details on the AIDAS system can be found in [95].

4.3 Metadata Model

In order to properly perform discovery activities, MIDAS adopts semantic-based metadata (profiles) to describe the properties and characteristics of involved entities, i.e., services and clients. The model defines a service as a "black box" that provides some functionalities to the external world, according to a declarative, object-oriented approach. With the concept of client, the model denotes any entity that exploits service functionalities

to achieve some kind of result. A client may be a human user looking for a specific service or a software component, like for instance a service broker, which searches appropriate single services to compose them into a complex service. MIDAS associates each client with a context including any information regarding the user that is relevant to the discovery process. For example, a human user's context may include the characteristics of the human person, e.g., the languages she is able to speak, as well as the technical characteristics of her device, e.g., 802.11b/g support.

MIDAS metadata model is designed to specifically support personalized user-centric discovery. In particular, as shown in the following sections, the model focuses its representation mainly on service/user/device capabilities and requirements. As a key feature, MIDAS metadata model provides fine-grained profile modularization to favor service discovery and selection effectiveness: the comparison between a user service request and service offers is not made over a complete service description, but over single service capabilities of interests. Device profiles further refine the searching scopes in order to provide users with only really usable and accessible services. Finally, MIDAS allows users to easily express priority preferences on requested service capabilities and on how to relax some user's requirements in the case exactly compatible services are not found. Users define priority preferences before service discovery is performed, to be relieved from the duty of manually selecting and ordering compatible services once they have been discovered. Requirement relaxation preferences can be optionally defined after discovery completion if no exactly compatible services have found to allow users to eventually retrieve alternative services of interests. Furthermore, in MIDAS preferences are explicitly collected from the user following an intuitive and user-based pattern, and they appear as a first-class metadata to guide service discovery.

4.3.1 Service Metadata

A service is described by a static profile and a dynamic profile. The static profile contains data that are relatively stable over time or do not depend on dynamic operating conditions, such as service name and functions. On the other side, the dynamic profile includes information that frequently change, e.g., location and state of the application.

Static Profile

The static profile consists of four sub-parts, namely: identification, service capabilities, service requirements, and service interface. *Identification* information provides information to name a service and to identify its location. Service *capabilities* are used to represent the functionalities a service provides and the way these functionalities are achieved, like for instance supported interfaces and communication protocols. For example, a service may exhibit the capability of performing software update via SSL from a remote site. A capability basically represents a logical unit of service functionality. It can be either a *core* capability, i.e., a functionality directly related to the service core activity, or a *functional* capability, i.e., an ability concerning properties that are not bound to the service activity, but describe how this activity is performed. For example, the remote update service has the core capability of "updating software" and the functional capability of "communicating over SSL". The latter is not specifically related to the service main activity since a software update service performs its task, i.e., updating software, independently from the presence of a SSL support. A capability example providing booking facilities for tourists is shown in Figure 4.2a. Service capabilities allow service providers or advertisers to describe their services in terms of offered functionalities. Users express their service requests in terms of desired service capabilities (see Figure 4.2b). To express service capabilities, we adopt a semantic approach. Therefore, we have defined a basic capability ontology. This

ontology needs to be integrated with the specific ontologies of the considered application, such as the news ontology in our example. Ontologies are modeled using OWL-DL [20].

Service *requirements* represent conditions imposed by the service in order to be accessed. In particular, a requirement is used to specify which capabilities a client wishing to access the service must exhibit. For example, a remote software update service normally requires that the client has a previous version of that software installed on her device. We distinguish between hard and soft requirements. Hard requirements are mandatory, while soft requirements may be described using a scoring function that determines the degree of importance for the requirement to be satisfied. MIDAS provides a requirement base ontology: specific requirement ontologies, such as a security ontology, have been developed by extending base classes and properties[2].

The third part of the static profile, i.e., the *Service interface* includes or points to the information needed to invoke the service, such as input/output description or the endpoint where to invoke the service. Let us note that such information might be provided according to different specifications, depending on the interface implemented by the service, like for example as a method signature for a Java object or as a WSDL profile.

Dynamic Profile

The dynamic profile describes service properties that might vary over time. In particular, the dynamic profile includes information about the state of a service, which represents the service operating conditions. Those conditions are dynamically retrieved via external information sources that provide, for example, information about service availability and load, or the average response time after service invocation.

[2]http://www.lia.deis.unibo.it/research/MIDAS/Ontologies/SecurityOntology.owl

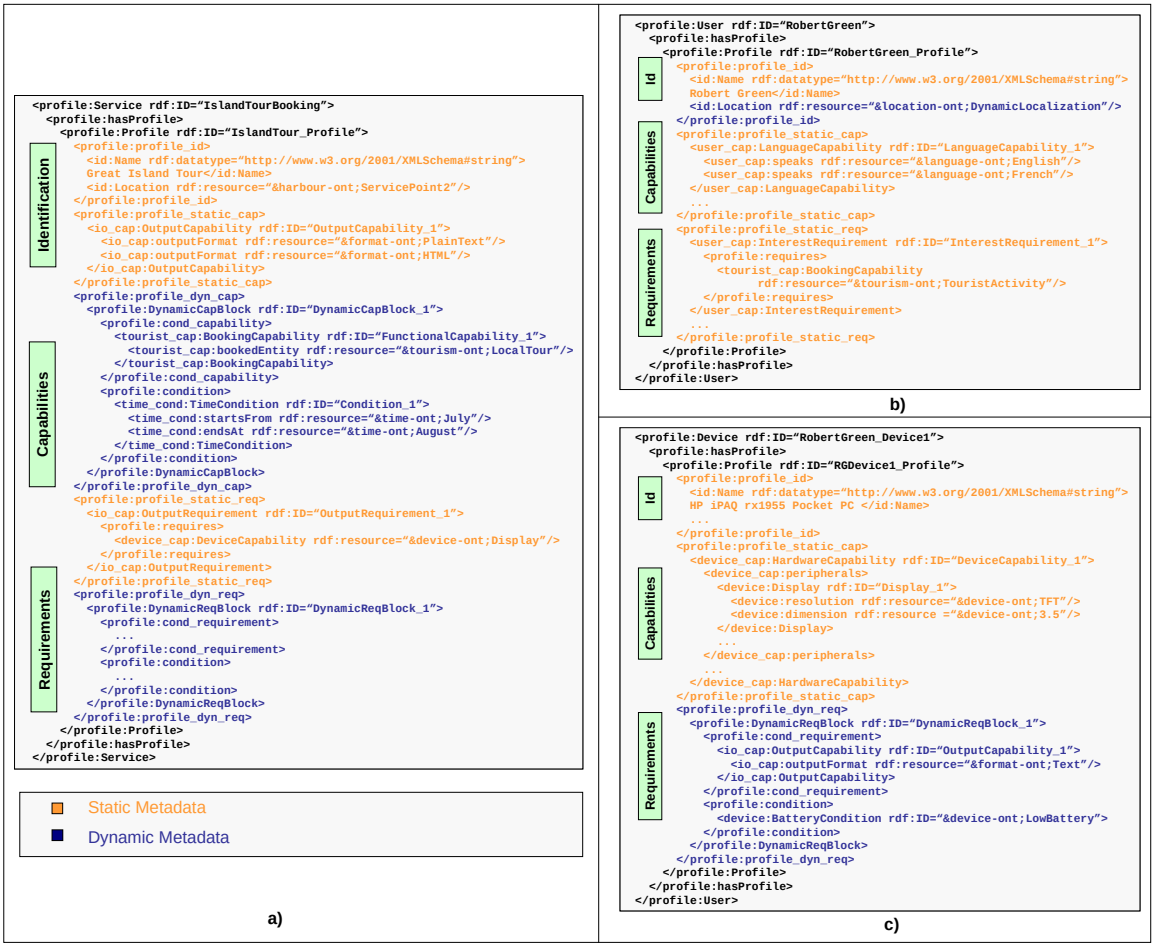

Figure 4.2: MIDAS service/user/device profiles.

4.3.2 User Metadata

Clients are described in terms of profiles and preferences. We herein focus on user *profiles* and *preferences*. The user profile is composed of dynamic and static metadata. Dynamic properties include, for example, locality and state. The static part of the profile contains three kinds of information: identity, capabilities and requirements. *Identification* information is needed to identify the user and may be expressed in various ways, e.g., an ID code, a string or an URL. *Capabilities* represent what the user is able to perform or to provide, such as the ability to understand a language. User *requirements* are conditions

imposed by the user that need to be always respected during service provisioning. Figure 4.2a shows an example of user profile.

Since a request for service may be expressed over several capabilities, and a capability might have multiple properties, we allow the user to establish a priority order among the various capabilities/properties by means of *priority preferences*. Such a preference enables to specify either an explicit priority index for capabilities/properties or to define a binary relation between two capabilities/properties. For example, a user can state that for a "newspapers online" capability it is more important that the "language" preference is respected rather than the "topics" one. Hence, the first property to be tested for matching will be "language" and a service that exhibits a good value for this property will be considered more compatible than another having a better value on the "topics" capability.

4.3.3 Device Metadata

Device metadata describe the technical characteristics and operating conditions of a user device. Similarly to service/user profile, the device profile includes static and dynamic metadata, and is composed of the identification part, the capabilities part and the requirement part. The identification part includes device category and type, as well as names and parameters that allow device identification within a network, such as a MAC address or a Bluetooth ID number. Device capabilities represent technical characteristics, supported functions and resource state, such as memory storage capability, secure socket layer support, and battery level. Finally, device requirements specify technical conditions that must hold for the device to properly access services and interact with other devices. For example, if a device is able to connect to another device only via Bluetooth, then Bluetooth connectivity represents a requirement for that device.

4.4 Middleware Architecture

The MIDAS architecture provides a a set of functionalities to support service discovery and selection based on user context information and user preferences as expressed in user queries. Figure 4.3 depicts MIDAS logical architecture, designed in two layers. The lower layer provides core facilities for service naming and registration. The upper layer components facilitate profile encoding, manage user contexts, identify proper discovery scopes, and provide personalized service views depending on user context.

In the AIDAS extension, MIDAS support for *discovery management* has been supplemented with an additional *configuration management* set of functionalities. In particular, those additional components provide needed facilities to allow each portable device to advertise provided semantic functionalities to co-located devices, e.g., reasoners, to discover, if needed because of resource limitations, locally available discovery management facilities, and to choose whether to download on-board or remotely access needed semantic services depending on device properties. For a detailed description of AIDAS configuration management features, we refer the reader to [95].

4.4.1 Discovery Management Services

The **Metadata Manager** (MM) provides support for the specification, modification, checking for correctness, installation and evaluation of different types of semantic metadata. MM provides templates to support the user in the task of specifying user/device/service profiles. The use of templates allows to ensure that metadata are encoded in the correct format, i.e., compliant to MIDAS profile ontology, while preserving non technical users from the burden of dealing with profile specification. Let us note that MM does not perform semantic reasoning, but only syntactic compliance checking.

The **Discovery Manager** (DM) is responsible for determining and maintaining

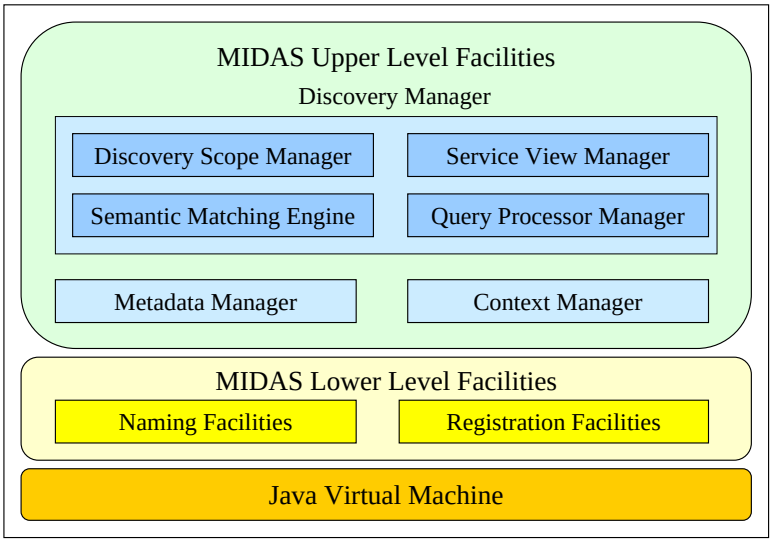

Figure 4.3: MIDAS middleware architecture.

the list of all services in the user's physical vicinity and of the specific services that are visible/accessible to that user on the basis of her context. In particular, among all services available in the user's network locality, DM selects the ones whose profiles are semantically compatible with user/device profile. The degree of compatibility between service and user/device profile is determined by applying the semantic matching algorithm implemented by PME to service capabilities and user/device requirements. For example, let us consider the case of a user that does not wish to use her credit card online. In this case, DM would not include in the user personal view any service that requires credit card payment. It may also happen the case of a user whose device does not support the Bluetooth protocol and who is therefore not enabled to access services provided via Bluetooth. In this case, DM would not include in the user service view Bluetooth-accessible services. The **Context Manager** (CM) is responsible for creating user contexts when MIDAS users initiate their discovery sessions, for monitoring changes in both created user contexts, e.g., in user profiles, and in relevant external environment conditions, e.g., the addition of new services, for

notifying changes to interested entities, and for updating user contexts accordingly.

The **Query Processor** (QP) is in charge of collecting and processing user requests for service. QP interacts with the user, via her User Proxy described in Section 3.2, to determine the required service capabilities and user preferences. In particular, QP is in charge of translated into a property restriction any value preference expressed by the user at access request time. Let us note that QP can also express disjunctive queries, which are specified by means of the OWL UnionOf construct.

The **Profile Matching Engine**(PME) is responsible for performing a matching algorithm between user/device requirements and service capabilities, taking user preferences into account. PME is requested to perform its algorithm in two cases, i.e., when DM needs to determine the list of visible services for a user, i.e., the list of services whose profile is compatible with user/device profile, and when QP needs to resolve a a specific user's query. In the first case, PME receives from CM user/device profiles and from DM the profiles of all locally available services. In the second case, PME interacts with DM to be provided with the list of user's visible services along with their profiles. In both cases, the static profile is used to perform direct matching, i.e., between user and device requirements and service capabilities. In particular, for each capability required by the user, PME verifies if the service profile contains one or more compatible capabilities. The matching algorithm is described in detail in the next section. The same algorithm is re-applied to the output of the direct matching to perform inverse matching, i.e., to match service requirements against user/device capabilities.

When determining the list of user's visible services, PME performs the matching algorithm on all locally available service, as provided by DM. On the contrary, in case a specific user request for service is being evaluated, PME can be differently configured. In particular, PME may either stop executing the matching algorithm at the first occurrence

of a compatible service, or perform the algorithm on each service visible to the user. In the first case, PME returns to CM the reference to a single service, while in the latter case it returns a list of services, ordered on the basis of the semantic compatibility results.

4.5 Prototype Implementation

We have developed a prototype implementation of the MIDAS middleware to be deployed in a wireless Internet scenario, i.e., a computing environment where wireless solutions extend the accessibility of the fixed Internet infrastructure via access points, working as bridges between fixed and mobile devices.

4.5.1 Naming and Registration Facilities

Recalling Figure 4.3, MIDAS is designed as a a layered architecture, whose lower layers provide core facilities for service naming and registration. In particular, MIDAS identifies services with Uniform Resource Identifiers; services can follow both announcement-based and pull-based approaches to advertise their availability; services can advertise their profiles either to a distributed directory or directly to interested parties in response to client requests. Naming and registration facilities have been implemented in two alternative versions. In the Jini-based version, the prototype relies on the presence of at least one centralized directory where providers can register their services to have them advertised via the Jini lookup protocol [3]. In case there is no centralized directory, we have implemented an alternative support based on the JXTA protocols to advertise and retrieve services within a peer-to-peer network [5].

In the next sections we focus on the upper layer facilities, which implement MIDAS key features.

4.5.2 Context-Aware Discovery Facilities

The Discovery Manager is in charge of determining a personalized view on services when a user starts her discovery session. DM includes a service registry component that allows service providers to advertise their services and users to look for them on the basis of a publish/subscribe mechanism. In particular, service providers can publish the description of their services using service templates that are provided by a dedicated MM instance integrated within DM. This MM instance only provides checking for syntactic correctness and compliance to the MIDAS profile representation model. Static service profiles are stored in the registry at start-up time and associated to a service identification code that permits to other middleware components, e.g., CM and PME, to reference them. Values of dynamic properties describing service state are not stored in the registry, but they are dynamically calculated at service access time by invoking appropriate methods on the service interface. DM exploits PME matching to discard services semantically incompatible with the user context, by exploiting context information. In particular, user/device requirements and service capability metadata are first used as PME input parameters to reduce the set of potentially compatible services. PME matching is re-applied to this subset of services with service requirements and user/device capabilities as input parameters for further context-based pruning. In addition, DM updates discovery scopes for on-going discovery sessions when CM notifies that services in other user contexts are newly added or have changed their profiles: DM applies PME matching again to those services to verify whether to include them in discovery scopes.

The Context Manager (CM) is responsible for creating and managing user contexts. CM can update user contexts according to different strategies: at pre-defined time intervals, or upon any context change detection, or upon explicit user request. The adopted strategy is decided at middleware configuration time and depends on several factors, from

user requirements to the desired trade-off between the need for fresh context information and the limitation of update overhead. CM has been developed by exploiting the context-awareness infrastructure and programming APIs of the Java Context Awareness Framework [19]. The current CM prototype implementation fully supports acquisition and management of static context information and of relatively simple dynamic information, e.g., time, location, and standardized monitoring indicators about device state, while we are still working on managing dynamic information harder to access in an open way, such as service load and expected network bandwidth/jitter.

The Query Processor consists of two main sub-components: the **Query Processing Engine** (QPE), i.e., the core processing module that performs the automatic translation of user/service requirements into required capabilities that can be processed by the Profile Matching Engine, and the **Query Processing Interface** (QPI), i.e., a user-oriented module that interacts with the user to define her service request and collect her preferences. QPI provides the user with a graphical interface to guide her during the specification of required service capabilities and preferences. Once the user has specified her service request and preferences, these data are forwarded to the QPE component, which translates them into OWL-based required service capabilities and related preferences. Then, QPE forwards these data to PME to be provided with (a list of) services that are semantically compatible with the user request.

The Profile Matching Engine performs matchmaking between offered and requested capabilities to determine the degree of semantic compatibility between user/device and service profiles. The details of the matching algorithm are provided in the following section. PME exploits the reasoning features of the DL-based reasoner Pellet [8] and the framework Jena [13]to acquire and manage ontologies.

4.5.3 Matching Algorithm

This section describes the matching algorithm we have implemented to perform preference-driven semantic selection of services.

As shown in Figure 4.4, the algorithm takes an offered capability and required capability and it returns the degree of semantic compatibility between them. Each capability is characterized by its properties. Let us note that offered capabilities are individuals, i.e., specific instances of a class, whereas requested capabilities are classes defined by restrictions, where restrictions on service properties are determined based on value preferences specified by the user. The algorithm works on one capability at a time. For each required capability, it is able to recognize three possible subsumption relations with the offered capability, namely: the offered capability may be an instance of the requested capability class (case exact), or an instance of a class that subsumes it (case subsumes) or an instance of a class that is subsumed by it (case plug-in) [80]. These semantic relations are determined by performing subsumption reasoning over the property values and class types of offered and required capabilities. In case of exact match for all service capabilities, the offered service is compatible with the user's request. In case the matching is not exact, compatibility is evaluated depending on value preferences: if there exists a user preference stating that the constraint over that property can be relaxed, a plug-in or subsumes case can be considered compatible.

In particular, let V_S be the vocabulary describing the service ontology specified in OWL-DL (*SHOIN(D)*). Let C_S be a class in V_S describing a concept capability. Let C'_S be a subconcept of C_S, i.e., C_S logically subsumes C'_S in the ontology interpretation. Let C''_S be a superconcept of C_S, i.e., C_S is logically subsumed by C''_S. Let the user requested capability be specified as a set of restrictions on the properties of class C_S.

- **case exact.** The offered capability is an instance of C_S. For example, the required capability class is of type Info_Capability, the offered capability is an instance of Info_Capability and the values of its characterizing properties satisfy the restrictions defined in the required capability class.

- **case plug-in.** The offered capability is an instance of C'_S. For example, the required capability class is of type Info_Capability and the offered capability is an instance of Newspaper_Capability, which is a subclass of Info_Capability, and the values of its characterizing properties satisfy the restrictions defined in the required capability class. Let us note that, being the offer more specific than the request, it might also happen the case that the offered capability is an instance of a subclass of the required capability (such as the News_Capability with respect to the Info_Capability), but one or more properties constrained in the request by means of value preference specification are not defined in the offer (such as "video broadcasting quality", which is not meaningful for newspapers). In this case, the algorithm might behave differently depending on priority preferences. If the considered property has low priority or is optional, the service might be considered compatible with the user's request.

- **case subsumes.** The offered capability is an instance of C''_S. For example, the required capability class is of type Info_Capability and the offered capability is an instance of a generic service capability, which is the superclass of all service capabilities, and the values of its characterizing properties satisfy the restrictions defined in the required capability class. Let us note that, being the offer more generic than the request, any property constrained in the request will have a value in the offered capability instance.

As discussed in the plug-in case, priority preferences are exploited to evaluate the degree

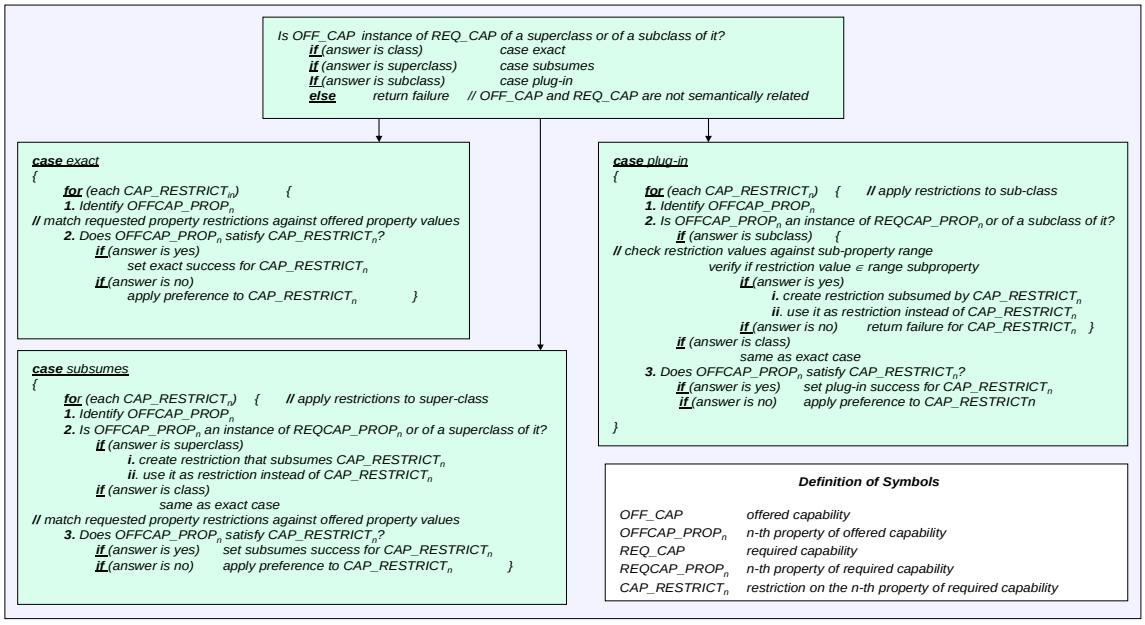

Figure 4.4: MIDAS semantic matching algorithm.

of relevance of a specific property in compatibility determination. In addition, priority preferences can be applied to determine the order according to which the algorithm should check compatibility of offered/required capabilities and properties. As a key feature of MIDAS, the prioritization of capabilities and properties allows to evaluate service properties according to the importance they assume in the user's request. This ability enables flexible, yet fine-tuned service filtering and provide the user with a personalized view on available services.

4.6 Case Studies

We have tested MIDAS in the design and implementation of a set of mobile applications that enable mobile users to access and retrieve services based on their current context. In particular, we have implemented the following prototype applications:

- A News Discovery Assistant (NDA) that enables mobile users to access and retrieve news from information services on a local basis. NDA retrieves information services available in the nearby of current user location, thus providing mobile users with news they might be interested in. NDA can be exploited, for example, to provide tourists with local news and information about ongoing events in the place where they are on vacation. Instead of manually browsing web portals and/or looking for printed information bulletins, tourists can define their personal interests and be automatically forwarded news and information on their device display [95].

- A service discovery assistant built on a MIDAS prototype, which we have deployed in a harbor scenario in the framework of the national Zefiro research project [23].

The following section will provide some implementation insights and performance evaluation about the MIDAS prototype we have deployed in the harbor scenario. Details about the NDA prototype can be found in [95].

4.6.1 The Zefiro Deployment Scenario

To describe how MIDAS components interwork during a discovery session, we consider the example of a user, Bob, who accesses services from a resource-limited device while on board of his docked boat.

Deployment Setting

We have deployed our MIDAS prototype components over both a fixed harbor computing infrastructure and devices on boats connected via IEEE 802.11 Access Points (APs). Each AP defines its network locality that includes services/infostations on local wired hosts and mobile wireless devices, playing the role of service clients or providers, within the AP

coverage area. For instance, one AP locality may include the `IslandTourBooking` service with the profile in Figure 4.2. At MIDAS deployment time, it is necessary to decide where to allocate middleware components. In the above scenario, we have decided to distribute all non-client MIDAS components on fixed hosts in the harbor network: there is one centralized directory for service registration, while one instance of any other server-side middleware component is allocated on fixed hosts in each AP locality. The used directory provides lease-based publishing with the event-based Java Naming and Directory Interface. About client-side MIDAS deployment, resource-rich devices have all MIDAS client facilities installed, whereas resource-constrained terminals only host lightweight stubs capable of forwarding discovery/service requests to dedicated proxies running on fixed hosts in the user AP locality. MIDAS proxies, implemented by specializing CARMEN-based mobile agents for the wireless Internet, called *mobile proxies*, [22], work on behalf of associated clients over the fixed network independently of possible temporary disconnections, maintain a copy of user/device profiles, and access services by coordinating with dedicated MM, CM, DM, and QPM instances.

MIDAS users interact with the server-side application via device-specific clients running on their wireless access devices. Client-side applications enable users to subscribe to MIDAS by filling in a form with user profile and to authenticate themselves to the service before starting any discovery session. When a user first accesses the service, MIDAS instantiates a shadow proxy in the domain where the user is currently attached. At service provision time, the clients are only in charge of forwarding user requests via QPI (and of visualizing the received service results) to (from) their responsible proxies.

Discovery Scope and Service View Creation

After docking his boat, Bob starts a discovery session. The stub on his device triggers MIDAS to generate a companion proxy, running on the fixed network and possibly migrating to maintain co-locality with Bob. The proxy maintains a copy of user/device profiles and generates its MM, CM, DM, and QPM instances. DSM is configured to retrieve the list of services (provided by either harbor information points or docked users) in the AP locality. After that, it coordinates with SME to produce Bob's initial discovery scope. Then, DSM commands SVM to start its work: SVM analyzes dynamic context conditions and creates the two tables for services included in/excluded from the view (see Figure 4.5). For instance, when checking whether the `IslandTourBooking` service is included in the view, SVM verifies the relevant static and dynamic conditions in Bob's/device/service requirements and capabilities, e.g., whether his device supports the visualization format for the service results that the proxy will forward to the stub (static) and whether the current date is between July and August (dynamic).

Discovery Scope and Service View Update

Let us suppose that another boat, e.g., Greg's yacht, is approaching the harbor with an on-board PC providing additional services, such as a shared repository offering high-resolution pictures and a blackboard service with indications/suggestions about visited bays. Via the registration facility locally installed, Greg's yacht registers the offered services in the MIDAS directory. The CM instance of Bob's proxy is notified of the new registration and Bob's DSM updates the personalized view accordingly. Then, via SME, Bob's SVM finds out that Greg's photo repository is not compatible with Bob's dynamic device capabilities because the device battery level is currently too low to sustain the possibly long downloading of high-resolution pictures. Therefore, SVM inserts Greg's repository in the table of services

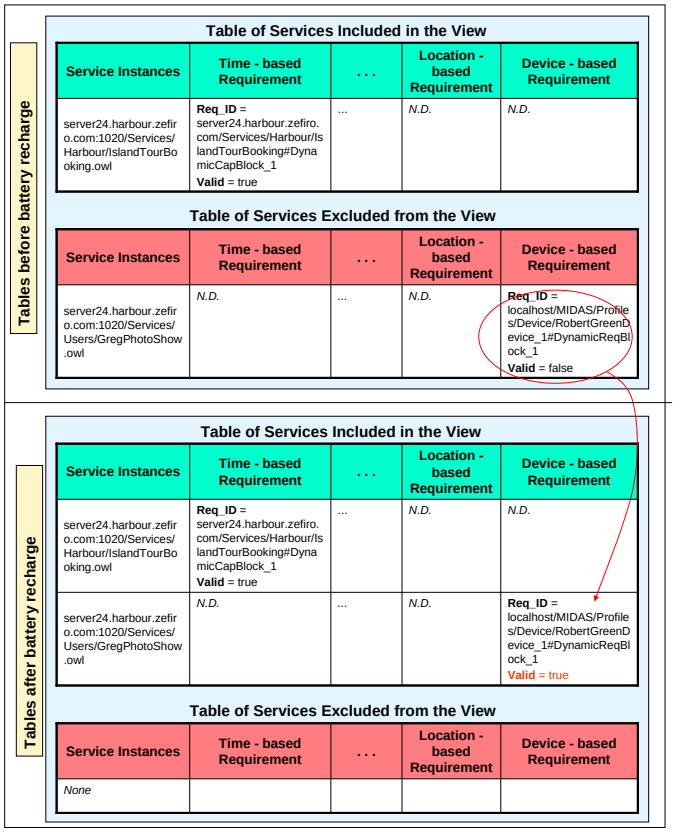

Figure 4.5: MIDAS tables for services included in/excluded from service view.

excluded from the view and annotates the dynamic requirement causing the exclusion. When Bob re-charges his device, CM senses the user context change and notifies SVM that re-evaluates the dynamic conditions associated with battery level (see Figure 4.5). information on their device display.

4.7 Evaluation

The exploitation of a context-aware semantic middleware for service discovery, such as MIDAS, introduces different forms of overhead, depending on both the deployment environment and the performance of the different middleware facilities, from profile parsing

to semantic-based query resolution.

We have extensively evaluated both the quality of our matching algorithm and the overhead introduced by the adoption of semantic metadata/techniques. We have considered a variable test-bed search space of 30 to 100 profiled services with requirement/capability ontologies following a hierarchical classification tree. In particular, the test-bed ontology nodes are in subclass relations; the ontology tree depth (maximum degree of requirement/capability specialization) is 4 and its breadth (multiplicity of requirement/capability related concepts) is 3. Each service has either one or two capabilities modeled in our ontology. Each user's request has a variable number of preferences, from 1 to 4, on a single capability[3].

To evaluate the quality of our matching algorithm, we have measured *recall*, i.e., the extent to which all relevant registered services are retrieved (by avoiding false negatives), and *precision*, i.e., the extent to which only relevant services are retrieved (by avoiding false positives) [25]. Being our matching algorithm complete with respect to our service ontology, its recall is optimal. This means that MIDAS is able to find all services whose capabilities have a semantic relation (as defined in Section 4.5.3) with user requested capabilities, according to the service ontology we have designed. Future work will be devoted to the exploration of heuristic-based pruning techniques to improve matching response times by possibly sacrificing completeness.

About precision, we have considered the case of services with two capabilities and user's requests with two preferences. We have then compared the number of MIDAS-retrieved services with the service set retrieved by Jini (by exploiting the Jini Technology Starter Kit, version 2.0, and representing service capabilities as Jini attributes). MIDAS has demonstrated to improve Jini precision of roughly 77% in the considered testbed. This

[3]Our test ontologies are available at http://www.lia.deis.unibo.it/research/MIDAS/OntoIndex.htm

outperform is due to the adoption of a service ontology that separately defines service capabilities and requirements. While Jini only allows to define and look for service attributes, MIDAS allows to define either capabilities or requirements with different meanings. Users requirements and preferences are only matched against service requirements, thus reducing false positives.

Finally, we have evaluated MIDAS query response time (between an explicit user request and the determination of discovery results). This test was executed on an AMD Athlon XP 1600 processor, equipped with 256 MB of RAM, running Windows XP Home Edition. PME was implemented using Jena 2 and Pellet 1.3 (on JDK 1.4.2). In our test-bed evaluation, services and middleware components reside on the same node. This test was carried considering a variable number of preferences on the same capability. With a test-bed search space of 100 services, the response time for a service query varies from approximately 9 ms, for a query defining only one preference, to roughly 12 ms, for a more complex query defining four preferences, as shown in Table 4.1. The query answering process basically involves four stages: parsing ontologies into a reasoner-compliant format, querying parsed ontologies on the basis of the user's request, performing reasoning over query results and ordering results depending on the degree of compatibility. We have evaluated the single contributions of each phase to the total response time. Our tests, whose results are depicted in Table 4.2, show that the most time-consuming activities are ontology parsing and querying, which are responsible for roughly 55% and 40%, respectively, of total query response time. Reasoning takes a very limited percentage of the total time (about 5%), while ordering time is negligible. It is worth noting that a significant variation on query response time might stem from variable network conditions. However, since such a variation might be difficult to evaluate and control because it depends on external conditions, we do not to consider it in our evaluations.

Semantic Matching Time (ms)			
Service Request Complexity (n. of restrictions)	Service Offer Dimension (n. of available service instances)		
	33	66	99
1	7511	7851	9173
2	7631	8472	9885
3	8041	9113	11627
4	8111	9144	11707

Table 4.1: MIDAS semantic matching time performance.

4.8 Related Work

Service discovery has always represented a crucial activity in the evolution and deployment of distributed systems. Discovery solutions for traditional distributed environments, such as Jini, CORBA or DCOM, typically relied on the assumption of a shared agreement among interacting entities about how to describe and to invoke a service. However, as this assumption cannot clearly be made in pervasive environments, traditional discovery solutions seem inadequate. On the other side, the Web service community has relaxed these assumptions, allowing to describe services in XML and to retrieve them essentially basing on keywords and fixed taxonomies, such as in the case of UDDI and its white-pages and yellow-pages mechanisms [6]. However, little support is provided to perform service discovery on the basis of service capabilities or user defined data. In addition, Web Services protocols lack of semantics causes service discovery to be very imprecise in case the user is not provided in advance with a syntactically defined description of the service features she can look for. Authors of [75] developed an augmented version of an UDDI registry that permits to attach user metadata expressed by RDF triples to common service descriptions. This allows to overcome the expressive limitations of the UDDI registry by

Single Contributions to Total Semantic Matching Time (ms)			
Semantic Matching Phase	Service Offer Dimension (n. of available service instances)		
	33	66	99
parsing	5728	5739	6169
query	1873	2774	4797
reasoning	500	620	751
result ordering	10	11	10

Table 4.2: Detailed time performance for a request with 4 restrictions.

adding different kinds of metadata, e.g., service ratings, functionality profiles attached to service sand semantic types attached to operation arguments. However, since inferencing is separated from discovery not to degrade discovery performance, this solution does not actually exploit the semantics of metadata as MIDAS does. In addition, unlike MIDAS, it is bound to a specific discovery protocol, i.e., UDDI.

In recent years, several research efforts have emerged to enhance service discovery and matchmaking, particularly by means of semantic-based technologies [42, 62, 70]. In [64], authors suggest that these efforts can be divided into two categories, i.e., special request language and query by example instance. To the former belong discovery solutions that adopt a special purpose query language, such as a SQL-like language for UDDI repositories. These solutions have the advantage of providing the user with several options for preference specification. MIDAS allows preference specification without requiring any additional language but simply exploiting its metadata model to describe both request and offer.

On the other side, "query by example" approaches adopt the same language for both service request and offer, describing the request as an instance of the ideal service. This requires to perform a matchmaking process to evaluate the similarity between the re-

quest and the offer instance. A well-known example of this approach is represented by the Semantic Matchmaker developed at Carnegie Mellon University [80]. From CMU Matchmaker we adopt the well-known semantic relations of exact, subsumes, plug-in. Other relevant pieces of work have extended these basic categories with similarity-based relations between services [65], additional logic-based relations [70] or potential/partial match relations [78], and implemented appropriate algorithms to compute and give ranking scores to compatibility and/or similarity between services. MIDAS is an integration framework that exploits semantic-based matching to provide the user with a personalized view on services. Its key feature is the adoption of semantic metadata to customize the discovery experience based on user's characteristics and preferences. Therefore, it does not focus on the issue of implementing powerful matching mechanisms, but rather on the integration of multiple features, such as semantic support configuration and semantic-based discovery, and on the central role of the user for the application. Similar considerations apply to our approach to preference handling, which is a potentially complex issue as shown by relevant existing work [18]. Our approach was intentionally kept simple in order to keep preference specification a manageable task for the mobile user.

MIDAS relies on its own service ontology, expressed in OWL-DL. We are aware that several languages for service description have been proposed, such as OWL-S [44], WSMO [85] and Meteor-S [102], to model both service interface (input/output) and service process workflow. These languages and the matchmaking algorithms that exploit them are generally focused more on input and output description than on service capabilities. Since MIDAS is built to support mobile users in service discovery, our ontology describes service characteristics instead of service inputs and outputs as we believe this is a more intuitive description for a human user. However, we are considering to provide support in MIDAS for other languages.

Authors of [25] propose a classification of various discovery tools basing on their *precision*, that is, the extent to which the tool retrieves only the items the services is interested in, and on their *recall*, that is, the extent to which the tool retrieves all the items the services is interested in. To achieve a good level of both precision and recall, they propose a process-based service model, which aims at capturing service behavior, i.e., what it does, as a collection of sub-activities. Thus they define a specific query language and describe process models in terms of entity-relationship diagrams. MIDAS approach differs from the process-based one in that it does not model the service internal activity, or its process model, but only its external interface, according to a declarative, object-oriented model of service. Another relevant difference in the logical approach to service matching is that MIDAS relies on subsumption reasoning and instance classification provided by description logic, i.e., OWL, while this solution executes variable bindings on clauses.

The discovery middleware proposed in [103] is the most similar to MIDAS middleware. In fact, both solutions address discovery issues for pervasive environments and both allow for the specification of preferences by the user. In particular, the proposed solution implements an ontology browser for mobile devices and allows users to express their preference directly over ontology diagrams. Even if the direct manipulation of ontologies may increase efficiency in preference specification, as it requires a lighter computation on the middleware side, we believe it has the disadvantage of leaving to non-experienced users the burden of dealing with ontologies that may not always be clear to understand. In addition, unlike MIDAS, the proposed framework does not provide a complete user profile mode and cannot therefore exploit its associated information to determine user context.

4.9 Ongoing Work

The MIDAS framework has been implemented in different application scenarios and extended into the AIDAS framework to manage access to semantic support services from portable devices.

We have started implementing MIDAS in the wireless Internet scenario that we consider the most significant deployment setting for our user-centric semantic-based discovery. Other implementations of MIDAS for mobile devices connected together with only low range protocols, such as Bluetooth, are currently under consideration. In the following, we use the term network locality to identify a LAN with 802.11b access points as a bridge between wired and wireless devices.

We are also aware that in open and dynamic scenarios it is necessary to take into account security risks arising when mobile users perform discovery of services whose providers are usually unknown or untrusted. For example, crucial security issues for service discovery in pervasive scenarios are protecting user privacy and controlling access to services. Therefore, we are working on a re-design of the MIDAS middleware to enhance the current framework with security features. In particular, since we believe that each service should be protected not only when accessed, but also when searched by a potentially untrusted user, our focus is on the issue of controlling access to services during the discovery process, i.e., *before* the service is actually invoked.

4.10 Chapter Summary

This chapter has presented our middleware architecture MIDAS, which exploits semantic techniques to perform context-aware discovery of services based on user context and requirements. The design of our middleware solution also tackles the challenge of mak-

ing viable semantic-based discovery to resource-constrained portable devices by providing an extended version of MIDAS, AIDAS, which support configuration of semantic facilities to portable devices. In particular, the chapter has shown MIDAS semantic metadata model to represent users, devices and services, and MIDAS semantic-based algorithm to match user requests against service offers based on user/service capabilities and requirements. The chapter has also provided implementation details about the prototype architecture and evaluated the prototype in a case study by discussing performance results.

Chapter 5

The Proteus Access Control Framework

The search for static security - in the law and elsewhere - is misguided. The fact is security can only be achieved through constant change, adapting old ideas that have outlived their usefulness to current facts.

William Osler

This chapter presents Proteus, a novel access control framework for pervasive environments based on context-aware semantic policies. The first section presents a motivating application example, i.e., a spontaneous collaboration scenario. Then, it describes Proteus context and policy model, and middleware architecture. The rest of the chapter provides details and evaluations about the prototype implementation by considering a case study related to the motivating scenario. Finally, relevant research work in the area of context-aware and/or policy-based access control is discussed and conclusions summarized.

5.1 Motivating Scenario

To point out some unique challenges in dynamic mobile environments, we have considered the spontaneous coalition scenario of a meeting occurring during a conference among members of different universities working on a common project. In the remainder of the chapter, we use this meeting scenario as a running example to illustrate the main access control challenges and our solution guidelines.

In this meeting scenario, each participant may wish to grant access to her resources to other participants, in order to enable cooperation and knowledge sharing. Access to personal resources must be regulated in order to protect them from malicious access or misuse. However, the specification of adequate access control policies in the depicted scenario presents us with several challenges. For example, the complete list of participants may not be known in advance or it may be modified just before the meeting starts or even during a meeting, thus making it infeasible to define access control policies based on the requestor's identity.

Even the role-based approach seems cumbersome in controlling access to cross-organizational resources, since role definitions and hierarchies might vary across parties, thus making their interpretation difficult outside the specific boundaries of each organization. A possible solution might be the creation of a common ad-hoc role for all meeting participants, to which each participant delegates her roles, so that others are able to access her resources [71]. However, since roles required to access resources have to be separately assigned by each participant to this ad-hoc role, inconsistencies may arise between the access rights of the different members, e.g., in the case of a member being allowed to access another member's resources, but not vice versa. Moreover, the activation/deactivation of such temporary roles represents a critical security issue.

In order to properly control access to resources, we claim the need for a more general and comprehensive approach that exploits not only identity and role information but also other contextual information, such as location, time, ongoing activities, etc. In particular, we believe that it may be advantageous for each participant to define the access control policies for his managed resources simply according to the current conditions of the requestor, the resource, and of the surrounding environment, i.e., the current resource context. For instance, in an informal meeting, access should be granted to those who are currently located in the same room where the resource owner is located, if they actually participate in the activity/project relating to the meeting, as long as current time corresponds to the time scheduled for the meeting.

Access control policies should be associated with the combination of one or more context conditions and users should be instantaneously granted/denied access to resources on the basis of those specific context conditions. The integration of access control with contextual information has two main characteristics. First, it is an example of an *active* access control model. Active security models are aware of the context associated with an ongoing activity in providing access control and thus distinguish the passive concept of permission assignment from the active concept of context-based permission activation. Second, the exploitation of context as a mechanism for grouping policies and for evaluating applicable ones simplifies access control management by increasing policy specification reuse and by making policy update and revocation easier. In fact, in subject-based access control solutions, the tight coupling of the identities/roles of principals with their permissions and with the operating conditions in the system to grant permitted actions requires security administrators to foresee all contexts in which each principal is likely to operate. In pervasive environments where principals are typically unknown and where contextual conditions frequently change, this traditional approach may lead to a combinatorial explosion of the number of

policies to be written, force a long development time, and even introduce potential bugs. The traditional approach, when applied to pervasive scenarios, also lacks flexibility. New access control policies need to be designed and implemented from scratch for any principal when new context situations occur. In a context-centric access control approach, instead of managing principals and their permissions individually, administrators define the set of permitted actions for each context. When a principal operates in a specific context, the evaluation process of his permissions in that context is triggered.

Another difficulty in dynamic collaboration scenarios is that it is impossible to define in advance all necessary policies for all possible situations. These environments should permit new policies to be dynamically and easily specified on demand as new situations occur as well as allow existing policies to be adapted to meet changing conditions. For example, let us consider the case of a meeting that continues beyond its originally scheduled end time. It is essential to ensure that meeting participants can continue to access each other's resources as long as the meeting is actually taking place. It is therefore necessary to adapt previous policies to reflect the new conditions of the meeting. In the absence of policy adaptation support, access to the policy owner's resources would be denied after the scheduled time, since the conditions that limit the applicability of the policy, specifically the condition concerning time, would be evaluated to be false. In a traditional approach, the policy owner would have to specify another policy to grant access to her resources after the scheduled end time of the meeting. However, this solution presents several disadvantages. First, the resource owner might not be the policy administrator of her resources, and might be unable to specify the policy when needed. In addition, the specification of ad-hoc policies is not a correct approach to policy definition because it does not favor clarity or traceability, thus complicating policy management. Finally, in such a case, efficiency and security might collide. If the policy owner specifies an access control policy that grants access to her

resources for a short time interval, e.g., ten minutes, she might possibly be forced to specify the same policy several times because the eventual end time of the meeting is not known in advance. Conversely, a policy granting access for a longer period might allow undesired access to the user's resources after the meeting.

This simple example demonstrates the need for a new approach to policy specification that not only defines policies based on context information, but also allows the seamless adaptation of policies depending on current context. In this example, we need to "instruct" the system such that, if certain context conditions hold, the context activating the policy is still considered active. Essential for policy adaptation is appropriate modeling of contextual information that enables the policy framework to sense and reason about the current situation. This ensures adequate access control even in changing and possibly unforeseen conditions.

Another important principle is the adoption of semantically-rich representations for policy definition. A semantics-based approach allows description of contexts and associated policies at a high level of abstraction, in a form that enables their classification and comparison. This feature is essential, for instance, in order to detect conflicts between policies before they are actually enforced. In addition, semantic techniques can provide the reasoning features needed to deduce new information from existing knowledge. This ability may be exploited by the policy framework when faced with unexpected situations to react in a contextually appropriate way.

5.2 Overview

Following the above outlined design guidelines, we have designed and prototypically implemented a novel policy-based access control framework, called **Proteus**[1]. Proteus exploits context-awareness and ontological technologies for the specification, evaluation and enforcement of access control policies [96].

In the Proteus access control framework the role of context exploitation for controlling access control is twofold. Drawing inspiration from the role-based access control (RBAC) model that exploits the concept of role as a mechanism for grouping subjects based on their properties [87], we state that, the same as with role, the concept of context can provide a level of indirection between entities requesting resource access and their permitted set of actions on requested resources. Instead of assigning permissions directly to the subjects and defining the contexts in which these permissions should be considered valid and applicable, a system administrator defines for each resource the contextual conditions that enable one to operate on it. When an entity operates in a specific context, she automatically acquires the ability to perform the set of actions permitted in the current context.

In addition, we consider context crucial for enabling policy adaptation. In pervasive environments the conditions that characterize interactions between users and resources may be largely unpredictable. Consequently, policies cannot all be specified a priori to face any operative run-time situations, but may require dynamic adjustments to be able to control access to resources. We use the term "policy adaptation" to describe the ability of the policy-based management system to adjust policy specifications and evaluation mechanisms in order to enable their enforcement in different, possibly unforeseen situations. In this scope, it is crucial to be able to represent the various operative conditions under

[1] http://www.lia.deis.unibo.it/research/Proteus/

which policies should be applied, i.e., the context, and to define the expected behavior of the policy framework on the basis of such context variations [97].

Another fundamental design guideline of our access control model is the adoption of an ontological approach using Description Logic (DL) to context/policy specification to enable context/policy classification, comparison, and static conflict detection. We also adopt a rule-based approach by exploiting Logic Programming (LP) to encode rules. In particular, our rules allow to specify policies based on context variables, whose value is unknown at policy definition time, thus enabling the efficient enforcement of policies defined over dynamically determined context values. Let us note that our work does not aim at providing a unifying logical framework for DL and LP, which have well-known logical mismatches in their foundation, but rather at combining the logical results obtained by means of their respective reasoning features [94].

5.3 Metadata Model

Proteus access control model is centered around the concept of context that we consider to be any characterizing information about the controlled resources and about the world surrounding them. We adopt a resource-centric approach to context modeling: contexts are associated with the resources to be controlled and represent all and only those conditions that enable access to the resources. Contexts act as intermediaries between the entities requesting access to resources and the set of operations that can be performed on these resources. Access control policies define for each context how to operate on the associated resource(s). In particular, access control policies can be viewed as one-to-one associations between contexts and allowed actions (see Figure 5.1. Drawing inspiration from Java protection domains [47], we call these contexts hereinafter as *protection con-*

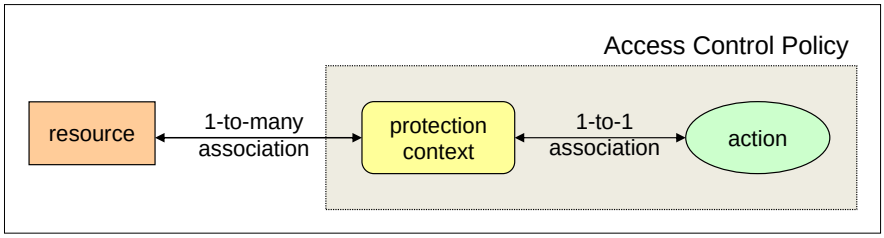

Figure 5.1: Proteus access control policy model.

texts: they provide users with a controlled visibility of the considered resource in terms of performable access actions on it (action view). Protection contexts are determined by the defined policies. Entities can perform only those actions that are associated with the protection contexts currently in effect (active context), i.e., the contexts whose defining conditions match the operating conditions of the requesting entity, requested resource, and environment as measured by specific sensors. All entities sharing the same active protection context share the same abilities to operate on the context-related resource.

5.3.1 Context Model

A protection context consists of all the characterizing information that is considered relevant for access control, logically organized in parts that describe the state of the resource associated with the protection context, such as availability or load (the resource part), the entities operating on the resource (the policy/resource owner and the requestor), such as their roles, identities or security credentials (the actor part), and the surrounding environment conditions, such as time, or other available resources (the environment part). A protection context is a set of attributes and predetermined values, labeled in some meaningful way and associated with desirable semantics [69]. Instead of a single value, an attribute could also define constraints for a range of allowed values. Let us note that an attribute value can be assigned to a fixed constant or can be a variable over a value domain.

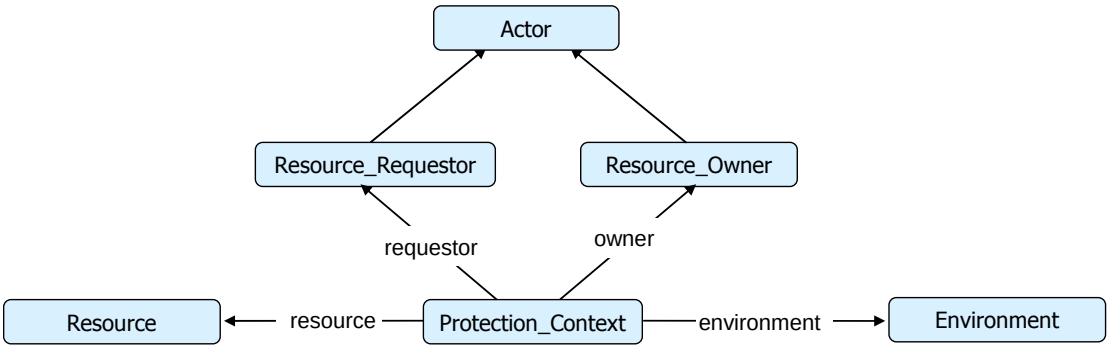

Figure 5.2: Proteus base context ontology.

The current state of the surrounding world is also represented in terms of attribute/value pairs where the attribute values represent the output of sensors (with the term "sensor" used loosely). For a protection context to be "in effect", the attribute values that define the current state of the world have to match the definition of the context (as given above). We adopt description logics (DL) and associated inferencing to model and process protection context data. In particular, we use Web Ontology Language (OWL) to formalize ontologies. A protection context is defined as a subclass of a generic context and consists of the resource, the actor and the environment context elements, as shown in Figure 5.2. Each context element is characterized by an identity property and a location property defining the physical or logical position of an entity. Single context elements are characterized by specific additional properties. Table 5.1 shows a DL-based protection context representation example related to the meeting scenario depicted in Section 5.1. This example assumes that each actor taking part to the meeting owns a set of resources that relates to the project/activity the meeting is about and shares these resources with the other participants. In particular, the protection context shown in Figure 1b grants access to these resources under certain conditions: the resources must be specifically pertaining the project discussed at the current meeting; the resource owner must be involved in the meeting's project as "project partner",

Meeting Context Specification
Meeting_Context ≡ Protection_Context ⊓ ∃owner.Meeting_Actor ⊓ ∃requestor.Co-located_Meeting_Actor ⊓ ∃environment.Meeting_Env ⊓ ∃resource.Current_Project_Resource
Meeting_Actor ≡ Actor ⊓ ∃is_currently_working_on.Current_Project ⊓ ∃located.Meeting_Space ⊓ ∃is_involved_in.Current_Project
Meeting_Env ≡ Environment ⊓ ∃time.In_Current_Meeting_Time
Current_Project_Resource ≡ Project_Resource ⊓ ∃is_resource_of_project.Current_Project
Co-located_Meeting_Actor ≡ ...

Table 5.1: Proteus protection context specification example.

must be currently work on the project-related set of resources, and must be located in the place where the meeting is planned to take place to guarantee that he is attending the meeting. The entities requesting access to resources must be involved in the project as "project partners", co-located with the resource owner, and currently working on project-specific resources on their devices. In addition, resources can be accessed when the time in the environment corresponds to the time scheduled for the meeting. Let us note that the core context ontology has been extended to model the specific meeting-related concepts. For example, a resource is associated with the project it relates to, an actor has attributes describing the project she is involved in or she is currently working on, and the environment time can be expressed in terms of scheduled events in an actor's calendar. The meeting ontology also explicitly defines the concept of "current event", which is an event or activity occurring at the moment of context and policy evaluation. In addition, we make use of a location ontology that is provided within the basic context model. Let us note that the use of DL in context modeling and reasoning has well-known benefits. For instance, considering protection contexts as classes and a set of sensor inputs (i.e., the current state of the world) as individuals, DL-based reasoning allows one to determine which protection contexts are in

effect by verifying which protection context classes the current state is an instance of, and to figure out how defined protection contexts relate to each other (nesting, etc.) [69]. However, DL-based reasoning may not always be sufficient. Our context-aware access control model needs more expressive context reasoning in order to be effective. On the one hand, we need to correlate contexts using not only class definitions (as in pure DL-based reasoning) but also property path relationships between anonymous individuals. For instance, in a meeting context we need to state that if the resource owner is located in a certain place and the resource requestor is located in the same place, the two are co-located. On the other hand, we need to bind the context attribute values to specific instances depending on application-specific context attribute/value relationships. For instance, to enforce the meeting-related policies, we must be able to determine, at each moment, what the actual current project is, so that the corresponding resources belonging to each actor are identified and protected. To overcome some DL-based reasoning restrictions we combine it with LP-based reasoning. In particular, we define two types of rules: context aggregation rules to support reasoning using property path relationships and context instantiation rules to provide OWL assertions for attribute values. For instance, the condition of co-location between two collaborating entities at a conference is expressed with an aggregation rule, whereas the condition of current project with an instantiation rule. Both types of rules are expressed according to the following pattern:

```
if context attributes C1...Cn   then context attribute Cm
```

that corresponds to a Horn clause, where predicates in the head and in the body are represented by classes and properties defined in the context and application-specific ontologies.

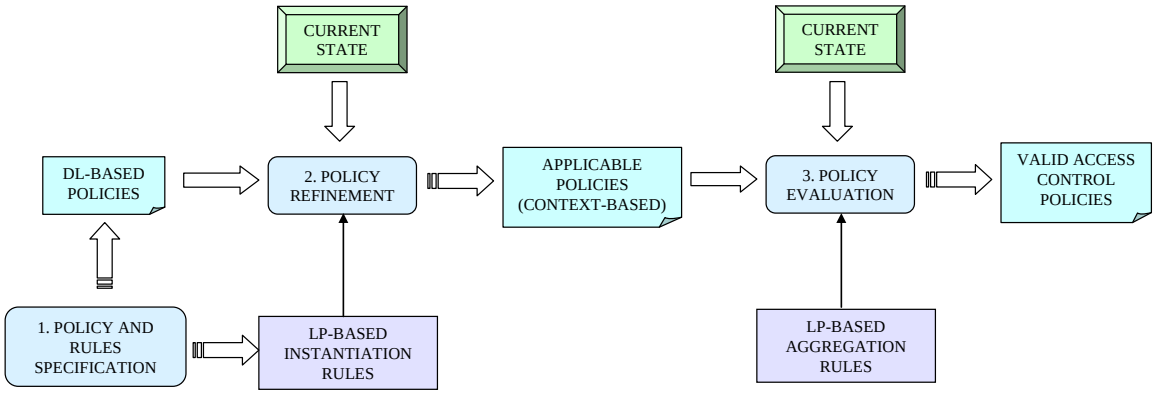

Figure 5.3: Proteus context-aware policy model.

5.3.2 Access Control Policy Model

Our policy model consists of three distinct phases (see Figure 5.3): policy specification, policy refinement, and policy evaluation. In the policy specification phase resource administrators specify OWL-based policies representing ontological associations between actions and protection contexts ontology definitions. Table 5.2 shows an example of a policy that controls access to the meeting resources.

The protection contexts may have attribute values assigned to constants or may be variables. In the latter case, attributes are assigned proper values by combining DL-based and LP-based reasoning over the context ontology and the context aggregation and activation rules. In particular, the output of LP rules is fed into the DL knowledge base to determine the value of each attribute given the current context. This means that OWL-based policies cannot be directly enforced into the system, but need to be further processed. By adopting an object-oriented terminology, OWL-based policies can be viewed as policy types: they define the actions that are allowed in a set of context types. In order to be enforced in the real world, policy types need to be transformed into policy objects that associate sets of actions with specific instantiated contextual conditions. In the *policy*

specification phase, administrators have to define aggregation and evaluation rules to enable effective enforcement and adaptation of OWL policies. For instance, in the meeting scenario an instantiation rule is needed to instantiate the current project attribute value included in the specification of the `Colocated_Meeting_Actor` class. The resource administrator could also define an aggregation rule to represent the "co-location" property as a relationship path based on the "location" property by means of variables. In the *policy refinement* phase, OWL policies are instantiated by adapting them to the particular state of the world, in order to obtain the set of applicable policies. In the *policy evaluation* stage, the protection contexts of applicable policies are verified against the current state of context elements as measured by sensors to determine the set of currently active policies. Let us note that the context-aware transformation process comprising of policy refinement and evaluation may be triggered by any resource context change, such as a new user requesting to access the resource or a significant change in the resource state, e.g., its location.

It is worth noting that our policy model adopts a combined approach to policy specification and reasoning. DL reasoning is exploited to perform static classification and conflict resolution of context and policy ontologies. LP reasoning is used to adapt the specification of OWL policies to the current state and allow their dynamic evaluation at access request time by means of appropriate rules. Adopting a combined approach allows us to benefit from the advantages of a pure ontology-based approach and those of a pure rule-based approach, both of which exhibit some limitations with respect to the definition and evaluation of policies and contexts [69, 94]. It is worth noting that our context model does not require the tight integration of the DL and the LP logical frameworks, which have well-known logical mismatches, but it is rather a combination of the two aiming at achieving more expressive description and reasoning capabilities about contexts and policies. In the following subsections we focus on the policy refinement and evaluation phases which

Colocated Meeting Actor Specification	
Meeting_Actor ≡ ∃is_currently_working_on.Current_Project ⊓ ∃is_involved_in.Current_Project ⊓ ∃colocated_with.Resource_Owner	
Instantiation Rules to be applied in case of an ordinary scheduled meeting	
Current_Meeting_Rule	Scheduled_Calendar_Slot (?x) ∧ Meeting (?x) → Current_Meeting (?x)
Current_Project_Rule	Current_Meeting(?x) ∧ Project(?y) ∧ meeting_on_project(?x,?y) → Current_Project(?y)
Instantiation Rules to be applied in case of a meeting prolongation	
Current_Project_Rule-2	Actor(?y) ∧ Last_Current_Project(?x) ∧ is_currently_working_on(?y,?x) ∧ Scheduled_Calendar_Slot(?z) ∧ Idle(?z) → Current_Project(?x)
Current_Meeting_Rule-2	Scheduled_Calendar_Slot(?x) ∧ Idle(?x) ∧ Past_Calendar_Slot(?y) ∧ Meeting(?y) Current_Project(?z) ∧ meeting_on_project(?y,?z) → Current_Meeting(?y)
Aggregation Rule to determine co-location	
Colocation_Rule	Actor(? x) ∧ Actor(?y) ∧ SymbolicSpace(?z) ∧ located(?x,?z) ∧ located(?y,?z) → colocated_with(?x,?y)

Table 5.2: Proteus policy specification example.

characterize our model and distinguish it from other state-of-the art related access control solutions [37, 98, 71].

Policy Refinement

Let us recall the meeting scenario to describe how policy refinement works. In the protection context of the meeting policy, shown before, the resource requestor property must belong to the `Co-located_Meeting_Actor` class that imposes that the resource requestor is co-located with the resource owner. Table 5.2 shows the definition of this context element, using a compact DL notation instead of OWL. Let us consider the restrictions applying to the properties `is_currently_working_on` and `is_involved_in`. These properties are restricted to a variable value, represented by the `Current_Project` class. This is an intrinsically variable value since the current project varies over time due to the changing activities of the resource owner and requestor, thus corresponding to different instances at

different time instants.

The defined context instantiation rules are used to determine the correct instance of the current project class at access request time. In particular, let us consider the first couple of rules shown in Table 5.2. The first rule establishes that, if the user's calendar shows a meeting for the current time, then that meeting has to be considered the current meeting. The second rule states that the project discussed at the current meeting is the current project. Once the facts about the user's calendar are inserted into the refinement fact base, the first rule is triggered and the inferred current meeting instance is used as a new fact to trigger the second rule. Then, the protection context is instantiated by re-writing it with the inferred context element values. For instance, if `ConnectingMe_Meeting` is scheduled on the user calendar, and `ConnectingMe_Project` is the corresponding project, then `Current_Project` is replaced by `ConnectingMe_Project` in the `Co-located_Meeting_Actor` specification. A new protection context is thus instantiated with the `ConnectingMe_Project` value and the corresponding policy generated with the instantiated protection context. The combined adoption of OWL policies and LP rules enables policy adaptation when needed. For example, let us suppose that the meeting has gone beyond the allotted time. Given this state, the first group of rules cannot be applied because there are no valid facts in their head. Therefore, a new set of rules has to be defined during the definition phase to cover the situation of an extended meeting. In particular, the first rule determines the owner's current project on the basis of her past and current activities, independently from her calendar schedule. For instance, if the last instance of current project (determined at pre-defined intervals or at access request time) was the `ConnectingMe_Project`, if the calendar does not show any event for the current time, and if the actor is working on the `ConnectingMe_Project`, then the `ConnectingMe_Project` is still the current project instance. The second rules checks for the last and the current

scheduling in the actor calendar. If there is no current event, and the last event was a meeting, and that meeting was about the current project (as determined with the first rule), then the last meeting is also the current one. In our example, the current meeting instance is the `ConnectingMe_Meeting`, as shown in Figure 5.3.

Policy Evaluation

We now describe the evaluation phase by using the same meeting scenario. When the current state of context elements, measured by sensors, is matched against the protection context of the meeting applicable policy, it is necessary to determine whether the protection context is currently in effect. During the evaluation phase the `Co-located_Meeting_Actor` definition of Table 5.2 is considered as well as the aggregation rule of Table 5.2 stating that if two actors are located in the same place (defined with the use of variables), they are co-located. Then, the resource owner's and the requestor's location are determined and inserted as facts into the evaluation fact base, which causes the execution of the co-location aggregation rule. Let us suppose that the requestor is co-located with the resource owner. In this case, a new fact is inferred that states that the resource requestor is co-located with the owner. This information is used to build the description of the current state of the world. In particular, an instance of the resource requestor element is created using the resource owner (which is known) as the value for the attribute co-location, and this instance of requestor is used in the protection context instance that describes the current state of the world. The created protection context instance is then compared with the protection context of the meeting policy by making use of ontology classification to recognize whether the former is an instance of the latter.

Instantiation Rules

Current_Meeting-Rule	Scheduled_Calendar_Slot (?x) ∧ Meeting (?x) → Current_Meeting (?x)
Current_Project-Rule	Current_Meeting(?x) ∧ Project(?y) ∧ meeting_on_project(?x,?y) → Current_Project(?y)
Scheduled_Slot-Rule	Calendar_Slot (?y) ∧ Calendar (?x) ∧ current_scheduling(?x,?y) → Scheduled_Calendar_Slot (?y)

ABox Assertions

(1) alessandra_Calendar: Calendar
(2) connectingMe_Project: Project
(3) connectingMe_Meeting: Meeting ⊑ Calendar_Slot
(4) <swapMe_Meeting, connectingMe_Project>: meeting_on_project
(5) <alessandra_Calendar, connectingMe_Meeting>: current_scheduling
(6) <alessandra, connectingMe_Project>: is_currently_working_on

semantic reasoning

Inferred Assertions

connectingMe_Meeting: Scheduled_Calendar_Slot
connectingMe_Meeting: Current_Meeting
connectingMe_Project: Current_Project

Table 5.3: Policy refinement example.

5.4 Middleware Architecture

The Proteus framework includes a middleware architecture that supports policy specification, semantic evaluation and enforcement based on current context conditions. Figure 5.4 shows the main components of Proteus architecture, namely: the Policy Specification Manager, the Policy Evaluation Manager, the Enforcement Manager and the Context Manager.

The **Policy Installation Manager** (PIM) is responsible for the setup, configuration and management of the Proteus systems. In particular, PIM provides support to load context and policy ontologies, and to install application-specific access control policies.

The **Reasoning Core** (RC) performs reasoning over context and policies to de-

termine currently active policies, according to the policy model described in Sections 5.3.2 and 5.3.2. In particular, by exploiting combined DL-based and LP-based reasoning, RC is able to determine which protection contexts and which policies are active given the current state. RC can be configured to perform its reasoning on a time basis, such as following a pre-defined schedule, or on an event basis, like for example in response to incoming access requests or to changes in relevant context dimensions, e.g., user location.

The **Policy Enforcement Manager** (PEM) is in charge of enforcing access control policies on protected resources. When a tentative access is performed on a resource controlled by Proteus, PEM intercepts the access action, collects relevant information about the action and interacts with RC to verify whether the access should be permitted or prohibited. Let us note that, in case there is no active policy controlling access to a specific resource, PEM decides whether to allow the access action based on the default behavior. PEM includes a set of local enforcers that are responsible for intercepting access actions within specific applications, such as a file access within a Java application or a remote desktop connection.

The **Context Manager** (CM) collects and manages context information from available context sources to provide PEM with data about the current state needed to perform policy and context reasoning. CM can update current state information according to different strategies: in case update is context-driven, any relevant change in context information triggers the acquisition of a new set of current state data. In case of a request-driven strategy, current state is re-evaluated upon receiving from PEM a request for access determination.

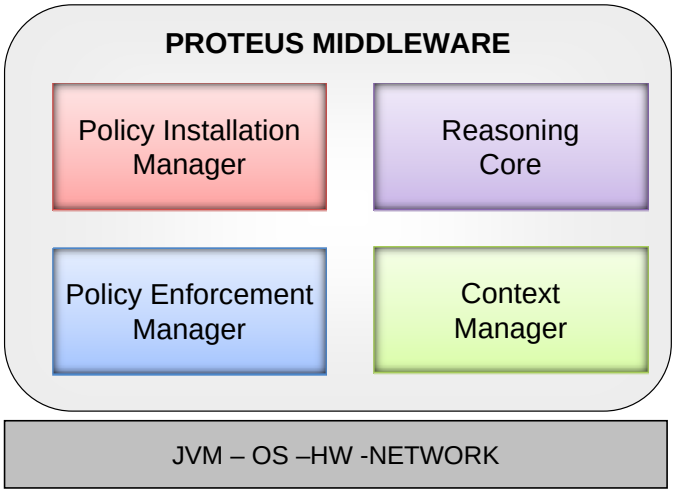

Figure 5.4: Proteus middleware architecture.

5.5 Prototype Implementation

We have developed a prototype implementation of the Proteus middleware architecture. Our deployment setting is a wireless Internet scenario, i.e., a computing environment where wireless solutions extend the accessibility of the fixed Internet infrastructure via access points, working as bridges between fixed and mobile devices.

In this section we provide some insights on the Proteus prototype implementation.

5.5.1 Implementation Details

Reasoning Core

The Reasoning Core (RC) is the key component of the Proteus middleware as it implements Proteus context-aware policy evaluation model. RC main subcomponents are the Ontology Manager, the Dependency Manager, the Reference Resolver, the ABox Generator, the Reasoning Engine and the Reasoning Core Controller (as shown in Figure 5.5, where arrows indicate control flows).

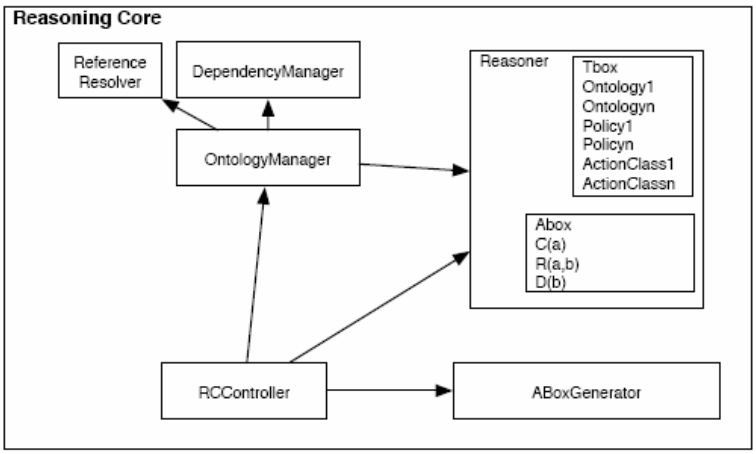

Figure 5.5: Proteus Reasoning Core main components.

The **Ontology Manager** manages ontology loading, installation and removal by coordinating with the **Dependency Manager**, which keeps track of dependencies between ontologies, such as import relationships, and the **Reference Resolver**, which actually retrieves ontologies from their identifying URIs, either locally or remotely.

The **ABox Generator** interacts with the Context Manager to be provided with assertions about the current state and to notify CM about newly installed context ontologies. Fact assertions about the current state are encoded in the form of (*subject, predicate, object*) triples, which can be either *asserted* or *retracted*. The ABox Generator is responsible for checking consistency of current state assertions, especially in case fact triples are retracted. Let us note that ABox Generator serves as a mediator between the actual reasoner and the context source, in order to avoid inconsistencies within the knowledge base used by the reasoner that would compromise the correct inferencing process. The ABox Generator can query CM to perform reasoning about a specific access request or subscribe to CM to be notified about changes in relevant context information. The **Reasoner** element realizes the wrapping of the actual inference engine, which in the current implementation is the Pellet

DL reasoner [8], accessed via OWL-API. Pellet has a preliminary implementation of a direct tableau algorithm for a DL-safe rules extension to OWL-DL. This implementation allows to load and reason with DL-safe rules encoded in SWRL, although some features of SWRL are not supported. By relying on Pellet combined DL and DL-safe rules reasoning support, the Reasoner is able to determine currently active protection contexts and policies according to the policy refinement and evaluation steps described in Sections 5.3.2 and 5.3.2. The Pellet instance also contains a repository for policy and context ontologies, stored in an internal format (the TBox), as well as a repository for asserted facts describing the current state (the ABox). Let us note that within an access control session, the TBox, which defines policy and context concepts, remain unaltered, while the ABox describing facts might change at each access request. Thus, to improve reasoning performance, the Reasoner also includes a local cache, where the TBox is stored in the OWLAPI format at policy installation time. This allows the Reasoner to only reload the ABox OWL ontologies at query evaluation time without having to parse again the TBox OWL context and policy ontologies, which can be directly accessed from the OWLAPI repository, thus saving time and bandwidth.

The **Reasoning Core Controller** is in charge of managing RC by coordinating ontology installation and removal, by keeping the state of the RC component monitored and by interacting with the Policy Enforcement Manager in case of incoming assess control requests. In the current implementation, RC is queried at access request time by PEM. Upon receiving a request for evaluation, the RC Controller coordinates with CM via the ABox Generator to be provided with up-to-date state information.

Context Manager

The Context Manager has been implemented by integrating with the context management and provisioning framework Contory [84]. Contory supports multiple context pro-

visioning strategies, namely internal sensors-based, external infrastructure-based, and distributed provisioning in ad hoc networks. In the current implementation, we only support infrastructure-based context acquisition by means of a context server. We are working on implementing context acquisition from internal sensors, where sensors, in a loose meaning, are software applications that, deployed on board of user devices, provide context information, such as a calendar showing user activities. Contory is queried via its SQL-like declarative language.

CM performs context processing to aggregate simple context information, as provided by the Contory interface, into higher level expressions, which are formalized according to a Description Logic formalism. In particular, as described above, CM produces OWL assertions, in the form of (*subject, predicate, object*) triples. Let us note that Proteus only defines a policy and a base context ontology, while application-dependent context ontologies must be added for any specific access action to be controlled. For example, to control access actions to a file via the File Transfer Protocol, it is necessary to define an FTP-specific context ontology including concepts such as file, transmission protocol, server and client. For any application-specific context ontology that is installed in the Reasoning Core, the corresponding context acquisition module must implemented and added to CM. In particular, this module must implement data processing functionalities according to the semantics encoded in the corresponding context ontology. At present we have implemented a simple context ontology describing access to a file within a Java application[2]. Let us note that no addition is required when new policies are installed in the system.

To allow the addition of context processing components, CM is built according to a modular structure. CM also provides a repository of available context acquisition module implementations, each one associated with the corresponding context ontology. CM can

[2]Our ontologies are available at http://www.lia.deis.unibo.it/research/Proteus/ontologies

provide context information either on-demand, such as in the case it is directly queried by the ABox Generator upon receiving an access request evaluation query, or according to an event-based strategy, for example in response to relevant changes in context information, as provided by context sources. In the latter case, CM supports a callback mechanism to let the ABox Generator register and be notified about variations in specific context data.

Policy Installation Manager

The Policy Installation Manager consists of two sub-components, namely, the Master Coordinator and the Policy Specification Tool.

The **Master Coordinator** (MC) is responsible for managing policy/context ontologies and for coordinating interaction between the middleware components. In particular, MC allows to configure the Proteus middleware to control access in a specific application setting by loading domain context ontologies into the PRM repository, and to install desired access control policies. In addition, for any new application-specific context ontology added to Proteus knowledge base, MC interacts with CM to add the corresponding context acquisition module. Once started the system, MC is in charge of coordinating PEM and PRM instances by dispatching access requests from PEM to PRM.

The **Policy Specification Tool** (PST) supports the definition of access control policies by providing tools for policy specification, modification, checking for correctness and installation. PST provides templates to specify OWL policies to relieve non technical users from the burden of mastering OWL complexity and to check policy specification compliance to Proteus policy and context ontologies. The Policy Specification Manager consists of a user-friendly interface that dynamically loads policy and application-specific context ontologies and present them to the user by means of structured representation. Our implementation of PST is based on previous relevant work in the area of semantic

policy specification interfaces, mainly due to the KAoS framework [101]. At present PST user-friendly interface only supports the specification of simple access control policies, while more complex ones can be defined by directly editing and loading OWL ontologies.

Policy Enforcement Manager

Policy enforcement is a strongly application-dependent issue. Therefore, the Policy Enforcement Manager includes different sub-components, namely Enforcers, Service Proxies and the Service Proxy Manager. **Enforcers** are responsible for monitoring controlled applications, intercepting each tentative access action and allow or deny the action based on the Proteus access control decision. Each Enforcer is implemented for a specific access action. We have currently implemented Enforcers to control access to files and communication channels (sockets) within a Java application by extending the Java Security Manager and re-implementing its methods [47].

Each Enforcer directly interacts with a dedicated **Service Proxy** (SP) instance, which mediates the interaction with the rest of Proteus middleware. Whenever a SP receives a request for access control evaluation from its corresponding Enforcer, it collects from the Enforcer relevant data about the request, such as the requestor's identity and the accessed resource, processes this information and passes it to the Service Proxy Manager. More in detail, the Service Proxy is in charge of translating request data encoded in an application-specific format into OWL assertions, according to the context ontology defined for that application.

Finally, the **Service Proxy Manager** is responsible for the installation and management of Service Proxies. Similarly to the case of CM, when a new application-specific context ontology is installed in the system, a dedicated SP must be implemented and added to PEM. In particular, when the SP Manager receives from the Master Coordinator a re-

quest for installation of a new Service Proxy, it looks for an appropriate SP instance in the SP repository. Once retrieved an appropriate one, it instantiates it into the system. Let us note that a Service Proxy is identified by means of a unique URI within the repository. At access request time, the SP Manager coordinates and dispatches Service Proxy requests for access control evaluation to the Reasoning Core via the Master Coordinator.

5.6 Case Study

We have tested the Proteus framework in the design and implementation of a collaborative application that supports dynamic interaction between mobile users by allowing the secure sharing of resources hosted on board of user devices. This section provides some insights on our prototype application by showing how to deploy, install and make use of Proteus context-aware access control features.

To describe how Proteus components interact during an access control session, we recall the spontaneous meeting scenario. In this scenario, each participant wishes to grant access to her resources to other participants according to certain context conditions. In particular, we consider the example of a user, Alice, who participates to the meeting and exploits Proteus to securely share a set of files with other users taking part at the same meeting.

5.6.1 Deployment Setting

As test-bed scenario for our spontaneous collaboration application prototype we have considered the case of a meeting about the ConnectingMe project taking place at the Faculty of Engineering. Connectivity is provided by IEEE 802.11-compliant access points (APs). In particular, each meeting room has one AP that provides wireless connectivity to all customers within the AP coverage area. Meeting participants are equipped with lap-

tops with IEEE 802.11b wireless cards running the spontaneous collaboration application. Participants' devices run all Proteus support facilities. For the sake of simplicity, each user device also hosts an instance of context server providing all raw context data, such as user location and activities, needed by Proteus to determine active policies. Policy and context ontologies reside on the lab server and are accessible via HTTP.

When a user first executes the collaboration application, an instance of all Proteus facilities and of the context server is created on his device.

5.6.2 Policy Installation

Before starting the application, it is necessary to install needed policy and context ontologies. The prototype includes a configuration file listing all needed ontologies, namely Proteus policy and context base ontologies, as well as the specific context ontology developed for the case of accessing documents in a meeting scenario. By changing ontology URIs in the configuration file, Proteus can be specialized to any application scenario. At start up, the Reasoning Core reads the configuration file, retrieves needed OWL ontologies, resolves reciprocal inter-dependencies and loads them into the system. More in detail, the Reasoner component parses both OWL-DL ontologies and SWRL rules into the OWL-API-compliant format, and stores them into the local OWL API cache, as shown in Figure 5.6. Then, the Reasoning Core notifies both the Context Manager and the Service Proxy Manager that new context ontologies have been installed in the system. CM in turn looks into its repository to retrieve the context acquisition module(s) associated with installed context ontologies. If a suitable implementation is found, it is inserted into CM. Similarly, the SP Manager searches in its SP repository a suitable Service Proxy for the specific application. If available, the SP is instantiated and a reference to the corresponding Enforcer implementation is provided to the user, who is only in charge of "installing" it into the application. Let us note

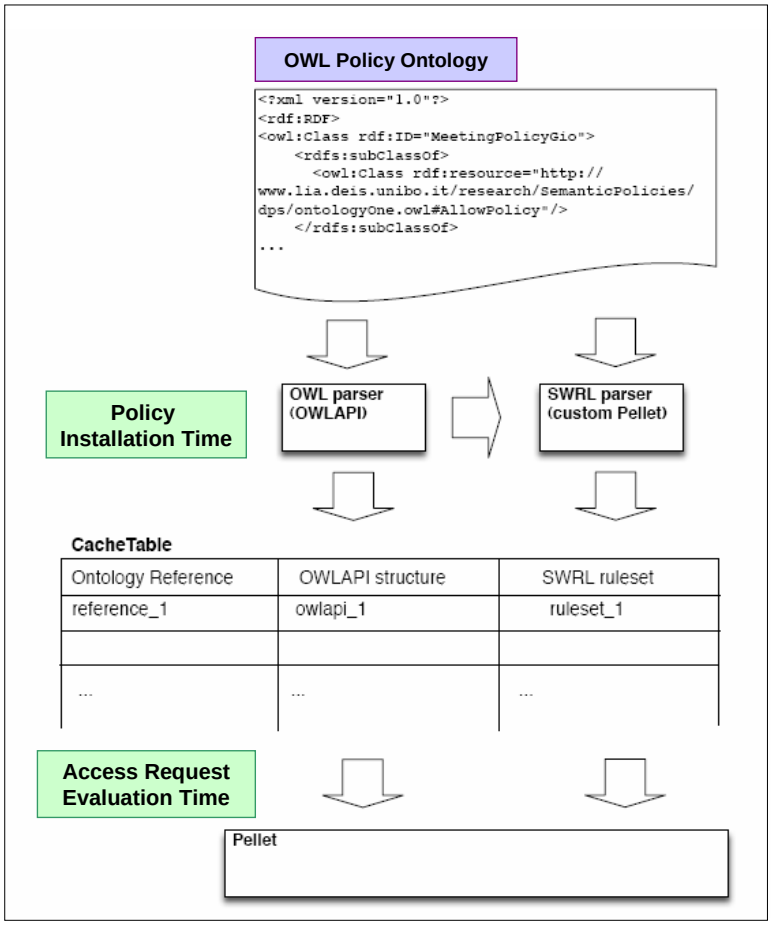

Figure 5.6: Policy ontology parsing and loading in the Reasoning Core.

that enforcer installation modalities might vary depending on the application. In our test example, it is required to embed into the code of the executable Java class some pre-defined lines of additional code, which instantiate a modified version of the Java Security Manager.

In case new policies are later defined and installed, the Reasoning Core performs the same checking and loading operations as above. However, since installed policies do not affect context semantics, CM and SP Manager do not need to be notified about this insertion.

5.6.3 Context-Aware Access Control Enforcement

In the following, we illustrate how Proteus intercepts and evaluates an access request by focusing on the main steps. We consider the case of two users, Alice and Bob, who are running a Java application on their laptop. Bob's application tries to access a file on Alice's laptop via a socket connection.

The tentative access is intercepted by Proteus via the Enforcer installed on Alice's device. The following steps summarize the control and information flow between the various middleware components. In particular,

1. The local Enforcer on Alice's device intercepts Bob's tentative access action and it forwards the access request to its related Service Proxy; the SP translates this request into a Proteus-compliant format and passes it to the SP Manager.

2. The SP Manager submits the request to the Master Coordinator, which fetches it into a request queue; when the request is retrieved from the queue, MC performs additional processing to translate it into a reasoner compliant format, and forwards it to the Reasoning Core.

3. The Reasoning Core first analyzes the request to derive which assertions over the context ontology are needed to properly determine currently active context and policies. Then, it queries the Context Manager to be provided with up-to-date assertions (ABox) about the current state for the previously determined concepts (TBox).

4. The Context Manager retrieves and provides the requested triple assertions.

5. Once obtained the assertions, the Reasoning Core performs combined DL and LP reasoning to determine (i) active protection contexts and (ii) active access control policies given the current ABox. By checking active policies, it determines whether

the access action to Alice's file is allowed.

6. The access decision is propagated back by means of callback mechanisms from RC to MC, to SP Manager and finally to the Enforcer via its SP.

Recalling the example discussed in Sections 5.3.2 and 5.3.2, let us suppose that the Meeting policy is currently active (see Figure 5.2). In this case, since Bob is allowed to access Alice's file, a positive answer is returned back to the Enforcer, thus permitting Bob to access Alice's file.

5.7 Evaluation

The adoption of a highly expressive policy model such as the context-aware Proteus model and the exploitation of semantic technologies might be responsible for the degradation of system performances. In this section we report some evaluations, both qualitative and quantitative, about the Proteus middleware.

The main elements that we consider in assessing the usability and efficiency of a mobile middleware framework are:

- *system availability*, including fault tolerance and load balancing
- *resource consumption*, mainly in terms of bandwidth and computational resources such as CPU and memory
- *system performance*, typically measured as system response time

In a mobile application scenario, the above mentioned aspects might be strongly influenced by network characteristics as well as by deployment settings. The Proteus framework does not define any specific deployment model. Our network reference model is the

wireless Internet, where fixed and mobile nodes are interconnected via wired-wireless connection. In this network setting, it is possible to choose among different options, from an infrastructure-based approach that provides Proteus facilities installed on fixed nodes, accessible by portable user devices, to a completely distributed and replicated approach that installs Proteus facilities on each user device. The former might be a good option for reducing resource consumption, but might decrease system availability due to the presence of a centralized element. The latter, while increasing resource usage, might ensure high availability since in case of fault both the access control system and its protected resources would be unreachable. Regarding performance, both approaches have advantages and drawbacks, which generally depend on the properties of both hosting devices and network connection.

After these premises, it appears more useful to attempt an evaluation over the peculiar features of the Proteus framework, i.e., the exploitation of context information and the semantic approach to policy definition and evaluation. In general, the response time to an access request, T_{RISP}, is due to different contributions:

$$_{TRISP} = T_{TRANS} + T_{QUEUE} + T_{CTX} + T_{REASON}$$

where T_{TRANS} is the round trip time needed to submit the request over the wireless network and receive the response back; T_{QUEUE} is the waiting time before the Reasoning Core processes the request; T_{CTX} is the time needed to retrieve context information about the current state; TREASON is the actual time employed to perform semantic reasoning over the request.

Hereinafter we focus on evaluations regarding the actual reasoning time. In the current implementation, this time is due to different contributions:

$$T_{REASON} = T_{PARSETBOX} + T_{PARSEABOX} + T_{LOAD} + T_{QUERY}$$

In particular, $T_{PARSETBOX}$ and $T_{PARSEABOX}$ represent the time needed to parse OWL ontologies into the OWL-API format, for the TBox and the ABox, respectively.

T_{LOAD} is the time to load OWL API ontologies into the Pellet native format, and T_{QUERY} is the actual time exploited to reason over the query. Thanks to the caching mechanism implemented in the Reasoning Core (see Section 5.5.1), the $T_{PARSETBOX}$ contribution is to be considered only once, at ontology installation time.

We have evaluated the variations of reasoning time T_{REASON} depending on the dimension of the TBox and ABox ontologies. An increase in the TBox ontology corresponds to new concept addition, while the size of the ABox reflects the number of concept instances in the application domain. For example, it is possible to define three instances of the concept of file. The following evaluations only consider DL-based reasoning.

Our tests have been performed on a Pentium43.0Ghz machine, equipped with 1Gb of RAM, running Linux kernel 2.6.17 kernel. The prototype is built using Pellet 1.5. Figure 5.7 shows the reasoning time variation with the increasing size of the TBox. In this case, the ABox linearly grows following the TBox by a multiplying factor of 3, i.e., for each concept in the ontology, there are three instances in the current state. The $T_{PARSETBOX}$ has not been considered since it is only relevant at system start up. Let us note that, in the case of nearly 100 concepts and 300 instances, the reasoning time keeps limited, around 450 ms. Figure 5.8 shows the reasoning time variation in the case of a fixed size TBox and a variable size ABox. This appears to be a common usage situation since context ontologies might be fairly simple, while the set of user resources to be protected might be rather large. With 65 concepts and 300 instances, reasoning time keeps below 400 ms, which is a compatible result with the intended use of the application. We have also evaluated combined DL- and LP-based reasoning. Probably due to implementation choices in the Pellet reasoner, reasoning time shows a strong dependence on both the number of rules and the number of variables defined within rules. Overall time performance is still not fully acceptable for a middleware whose purpose is dynamically controlling access to resources. We are working

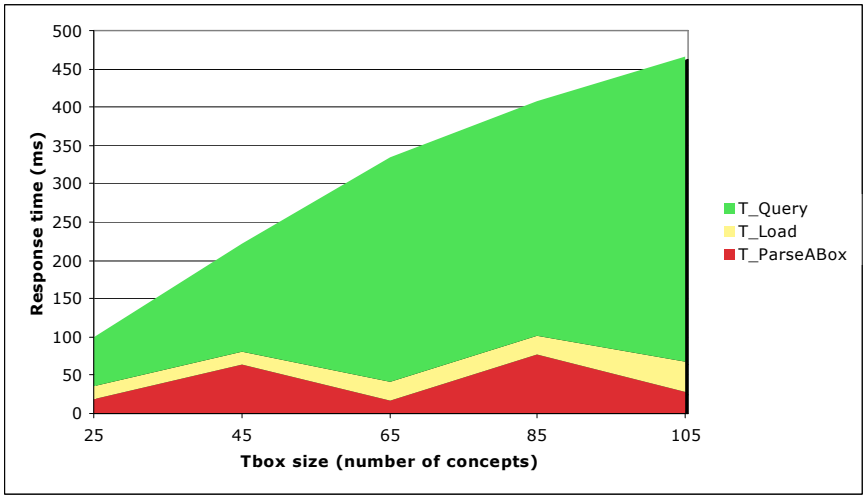

Figure 5.7: Reasoning time variation with TBox dimension.

to understand the reasons for this unsatisfactory Pellet performance and possibly plan to adopt a different reasoner.

5.8 Related Work

Several research efforts have addressed the issue of access control in dynamic environments. We do not intend to provide a general survey of the state-of-the-art access control solutions in dynamic environments, but only to focus on the research that either integrates context-awareness and semantic technologies into access control policy frameworks for pervasive environments or addresses access control issues in similar coalition application scenarios. Considering context explicitly for access control is a very recent research direction with only few context-dependent policy model proposals.

The importance of taking context into account for securing pervasive applications is particularly evident in [37] that allows policy designers to represent contexts through a new type of role called environment role. Environment roles capture relevant environmental con-

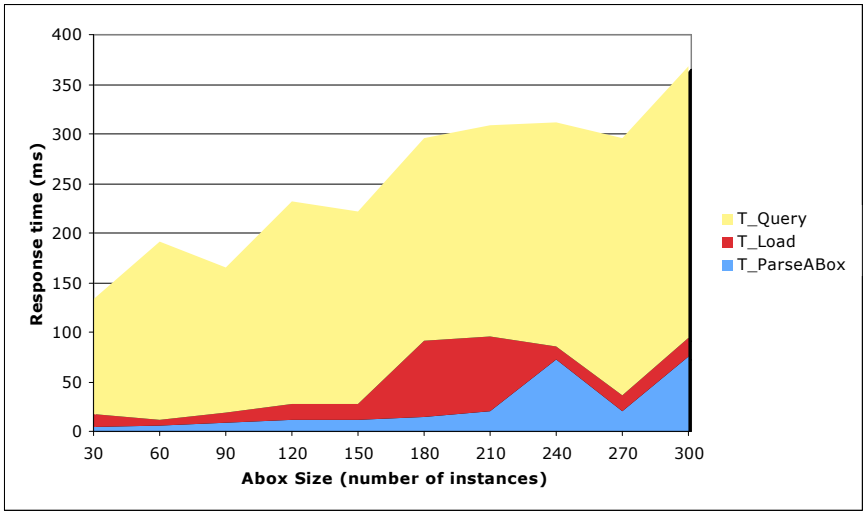

Figure 5.8: Reasoning time variation with ABox dimension.

ditions that are used for restricting and regulating user privileges. Permissions are assigned both to roles (both traditional and environmental ones) and role activation/deactivation mechanisms regulate the access to resources. Environmental roles are similar to our contexts in that they act as intermediaries between users and permissions. However, because environmental roles are statically defined in terms of attribute-constant value pairs their evaluation cannot provide support for policy adaptation as in our proposed semantic context-aware approach. In addition, differently from our approach, in [37] there is no integrated support for representing at a high level of abstraction and reasoning about environmental roles and policies. By focusing on access control in spontaneous coalitions in pervasive environments, [71] proposes a delegation-based approach, where users participating to a communication session can delegate a set of their permissions to a temporary session role, in order to enable access to each other's resources. In particular, one end-point user assigns the session role to the entities he is willing to communicate with. Contextual information is used to define the conditions that must hold in the system in order for the assignment to take

place, thus limiting the applicability scope of this process. Only a limited set of contextual information can be specified and no semantic technologies are exploited to represent nor the session role nor the delegation context constraint. In addition, security problems may arise whenever an entity delegated to play the session role leaves the communication session. In fact, unless the user explicitly states she is leaving the session, there is no way for the framework to be aware that the session role must be revoked for the departing user. The importance of adopting a high level of abstraction for the specification of all security policy building elements (subjects, actions, context, etc..) is starting to emerge in well-known policy frameworks, such as KAoS and Rei [98, 94]. KAoS and Rei represent, respectively, significant examples of DL-based and LP-based policy languages. In particular, KAoS uses OWL as the basis for representing and reasoning about policies within Web Services, Grid Computing, and multi-agent system platforms [101]. Contextual information is represented as ontologies and is used to constrain the applicability of policies. The KAoS approach, however, relying on pure OWL capabilities, encounters some difficulties with regard to the definition of certain kinds of policies, specifically those requiring the definition of variables. Rei adopts OWL-Lite to specify policies and can reason over any domain knowledge expressed in either RDF or OWL [60]. A policy basically consists of a list of rules expressed as OWL properties of the policy and a context represented in terms of ontologies that is used to restrict the policy's applicability. Though represented in OWL-Lite, Rei still allows the definition of variables that are used as placeholders as in Prolog. In this way, Rei overcomes one of the major limitations of the OWL language, and more generally of description logics. i.e., the inability to define variables. On the other hand, the choice of expressing Rei rules similarly to declarative logic programs prevents it from exploiting the full potential of the OWL language. In particular, the Rei engine is able to reason about domain-specific knowledge, but not about policy specification. Our policy model shares some commonalities

with regard to context/policy representation with both KAoS and Rei, but differs in how it deals with context. Our approach considers context as the primary basis that allows one to deduce which policies apply to a subject acting in the system whereas KAoS and Rei, similarly to traditional approaches, exploit context to build filtering mechanisms for policy applicability.

5.9 Ongoing Work

Early results about Proteus seem promising and are encouraging our work to improve the system. In particular, we are developing new context ontologies, as well as implementing enforcers and context acquisition modules to provide support for the exploitation of Proteus in new application scenarios. We are also planning to enhance our context model with Quality of Context (QoC) parameters that are taken into account during the reasoning process. In fact, in a framework like Proteus, which takes access control decisions based on context information, the notion of QoC is connected with that of *risk* since variations in the QoC level lead to variations in the overall security ensured to the system. Along a different direction, we are investigating the issue of unsatisfactory performance of DL and LP combined reasoning with the objective of providing some optimization. Finally, we are planning to reimplement the Reasoning Core by exploiting Pellet latest version, which includes support for incremental reasoning.

5.10 Chapter Summary

This chapter has presented the access control framework Proteus. Proteus key features are its context-aware policy model and its semantic approach to policy specification and evaluation. These features allow the dynamic adaptation of access control policies in

response to context changes, frequent in pervasive environments. In particular, the chapter has shown the Proteus context model and semantic context-aware policy model, which adopts a combined representation and reasoning approach based on DL ontologies and LP rules. The chapter has then described Proteus middleware architecture, has provided implementation insights about the prototype implementation and evaluated the prototype in a case study by discussing performance results. Finally, significant related work has been reviewed and compared with our system.

Chapter 6

The SAMOA Mobile Socially-Aware Framework

As we learn to design calm technology, we will enrich not only our space of artifacts, but also our opportunities for being with other people.
Mark Weiser and John Seely Brown

This chapter describes our middleware-based approach to support anywhere and anytime social network creation, called called Socially-Aware and MObile Architecture (SAMOA). It first presents an application scenario with the aim of outlining the main design guidelines we propose for social computing middleware-level solutions. Then, SAMOA metadata model and middleware architecture are presented, with particular attention to social network management model. The chapter also provides implementation details about the prototype system and shows its usability in a viral marketing application built on top of SAMOA. Experimental results are shown to assess the framework usability in the target scenario. After discussing relevant related work, the chapter provides some insights on

ongoing work and presents our conclusions.

6.1 Motivating Scenario

To point out the emerging requirements of social network management in pervasive environments, we start by discussing a *viral marketing* scenario. Viral marketing and viral advertising refer to marketing techniques that exploit preexisting *social networks* to produce increases in brand awareness, through self-replicating viral processes, analogous to the spread of pathological and computer viruses [76]. Viral marketing is a marketing phenomenon that facilitates and encourages people to pass along a marketing message voluntarily. Viral promotions may take the form of video clips, interactive games, e-books, brandable software, images, or even text messages. Marketing-related information spreading, such as commercial promotions, is based on a word-of-mouth mechanism: a user forwards a promotional message to all users in his social network. Information spreading can be also restricted according to market segmentation criteria that consider several customer-related information, e.g. their age, gender, education level and so forth.

Let us consider the case of a a shopping mall, where several vendors might wish to advertise discounts to potential customers that are currently visiting the mall. For example, a book vendor might wish to exploit the available wireless connectivity to forward promotion messages to the mobile devices of customers that are currently located in the bookshop, as well as to those of all customers that visited the shop and are still inside the shopping mall. In particular, the book vendor might define the target market segment by describing a set of attributes that characterize the potential customer, such as a certain age and level of education. Once a customer has received a promotional message on his portable device he might forward the promotion to users located in the bookshop, or to users inside the

shopping mall that are interested in purchasing books. Let us note that message forwarding is based on current user activity and location, and on the impromptu encounters with other users that are performing similar activities in the same place.

In the depicted scenario, a crucial issue to ensure the effectiveness of the word-of-mouth-based spreading effect is how to dynamically determine the set of potential customers to which the promotional message should be forwarded. This set of mobile users can be thought as a *social network*: the book vendor might define a social network including all his target customers, while each client visiting the bookshop might define a social network by grouping co-located users sharing the same interest about books, as well as the current activity of shopping. Therefore, proper social networking applications are needed that are able to exploit the wireless connectivity provided by user mobile devices to dynamically build social networks based on similarities between users, such as overlapping interests and preferences, age, gender and education, but also same location and current activity.

Most existing social networking solutions have been developed by riding the wave of the World Wide Web, which has made easily available a massive amount of data about users, for example from blogs, newsgroups and chat rooms. which can be used to extract significant social networks of interests. However, Web-based social networking solutions generally rely on the assumption that social relations can be established independently from physical places, and tend to promote a separation between physical places and the social spaces where interactions occur.

Technology advances in wireless networks and the increasing diffusion of portable/ wearable devices with both fixed and wireless connectivity offer a unique opportunity to further improve social networking services and to extend their scope of applicability. The possibility of ubiquitous computing of being connected anytime and anywhere enables serendipitous social encounters between proximate users with common interests and the formation of

ad-hoc spontaneous social networks on demand, anywhere and anytime [36]. For example, in the viral marketing scenario, users visiting the bookshop could exploit their wireless-enabled portable devices to be notified by the book vendor about available commercial promotions. Ubiquitous technologies promote a focus shift from virtual social spaces to physical social spaces and permits to re-establish the connection of social networks to physical spaces. The new underlying assumption is that user proximity and physical places affect and influence social behavior in many ways [58]. Physical proximity increases the likelihood of forming impromptu social relationships. In addition, physical places can act as social filters for people. People tend to go only in places that provide activities of interests where they are also likely to share common characteristics with other co-located people. For instance, only people interested in books are likely to visit the bookshop.

Along these directions several prototypes of social networking systems have recently emerged that exploit not only social preferences, but also co-location and/or reciprocal proximity of individuals as key design principle for guiding social network composition/management strategies and for restricting the scope of interactions among social network members. However, to realize full potential of anywhere and anytime social network computing several difficult technical challenges must be still addressed. In particular, because of the impromptu and transient nature of ubiquitous interactions another main challenge is to develop solutions able to extract social networks autonomously and transparently from users by minimizing user intervention. Achieving anytime and anywhere social network computing requires also shared and interoperable vocabularies for modeling location/entity characteristics to avoid inconsistent interpretations typically arising in open and heterogeneous ubiquitous environments.

We claim that the success of anytime and anywhere social network computing depends on the design and development of middleware-level solutions that integrate all

social network management facilities. Novel middleware-level solutions are therefore needed to allow social network application designers to focus only on application logic requirements while being relieved from the burden of addressing low-level social network management details, thus significantly simplifying and speeding-up application development. We claim that the design of middleware-level solutions for supporting anytime and anywhere social network computing should follow novel design guidelines.

A primary design guidelines is *context-awareness*. In line with the proposals in the field it is necessary to compose social networks on the basis of user location and/or reciprocal proximity. In addition, the visibility of profile information describing user attributes and social preferences should be also considered to further restrict the scope of interactions among co-located users sharing only common interests, activities and goals. The middleware should provide integrated support for context modeling and acquisition and for context-aware social network extraction.

In addition, because of the impossibility to make a-priori assumptions about the way user contexts are described in an open and dynamic deployment scenario, such a as the ubiquitous one, the other emerging design guideline we propose is the adoption of *semantic languages*. The main potential advantages of semantic technologies is that they permit a formal representation of user context properties at a high level of abstraction. On the one hand, that enables automated reasoning on context representations. On the other hand, it facilitates interoperability between entities that may wish to interact even if statically unknown. In particular, social network extraction can primarily benefit from a semantic-based approach: traditional social network discovery and extraction queries based on the exact matching of attribute patterns/keywords are likely to often fail in ubiquitous environments because users typically cannot have total/partial knowledge and agreement about needed service identifiers.

6.2 Overview

Our proposal is a middleware-level solution, called **S**ocially-**A**ware and **MO**bile **A**rchitecture (**SAMOA**)[1], for anytime and anywhere social application provisioning that integrates a set of common management facilities, e.g., for user location/proximity tracking, for creating and managing location-dependent SNs personalised to user's social preferences and for propagating the visibility of SNs up to the application level [28].

SAMOA is a single reusable middleware to dynamically instantiate and deploy depending on social computing application-specific requirements and execution environments. To enable the creation of SNs that reflect the reality of social interactions in ubiquitous environments, SAMOA provides integrated support for context modeling, acquisition, reasoning and for context-aware SN extraction, where context information includes user location and/or reciprocal proximity, user attributes, motivations, attitudes, activities and social preferences [81].

Another key distinctive feature of the SAMOA technological infrastructure is the adoption of a semantic-based modeling approach to user/location characteristics and a semantic-based social matching algorithm to infer relations among co-located individuals. Semantically rich representations of context information, such as user location and characteristics, allow to define context descriptions at different levels of abstraction and enable reasoning about both the structure and the properties of location and SN entities. In addition, emerging ontology standards, such as RDF and OWL, allow interoperability between possibly unknown users that may wish to establish a social interaction.

[1] http://www.lia.deis.unibo.it/research/SAMOA/

6.3 Metadata Model

The SAMOA framework supports the creation of anytime, anywhere semantic context-aware social networks-that is, the logical abstractions that group together mobile users who are in physical proximity and share common affinities, attitudes, and social interests. In particular, SAMOA lets mobile users create roaming social networks that, following user movements, reflect at each instant all nearby encounters of interest.

SAMOA roaming social networks center on a user (the ego user), and are based on two kinds of context visibility:

- *place* visibility (place awareness, that is, the visibility of the user's physical place
- *profile* visibility (profile awareness), that is, the visibility of place or user requirements and characteristics.

Place visibility restricts the discovery scope for social-network extraction to entities in the same place as the ego user. The visibility of user or place profiles further refines the discovery scope to create personalized social networks. In addition, SAMOA models and represents context data in terms of semantic metadata (profiles) and exploits semantic-matching algorithms for analyzing profiles and inferring potential semantic compatibility.

6.3.1 Social Network Management Model

The SAMOA social-network management model defines three management roles:

- *Managers* are the mobile ego users interested in creating social networks. They're responsible for defining the discovery scope boundaries of their social network and the criteria guiding its extraction.

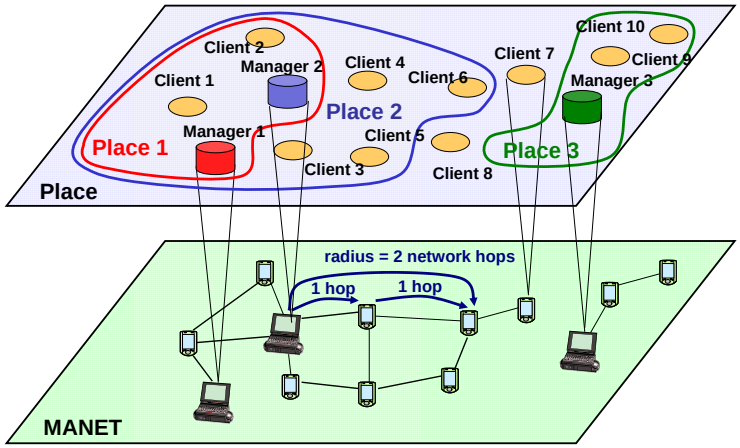

Figure 6.1: An example place mapping of SAMOA onto a mobile ad hoc network.

- *Clients* are users located within the discovery scope boundaries and are eligible to become members of the manager's social network.

- *Members* are users affiliated with a social network.

Each mobile user can play all roles. The manager role can be covered by a human or by a software component acting on the ego user's behalf. In SAMOA, social-network management is based on the concept of place, which lets us establish well-defined discovery scope boundaries. As Figure 6.1 shows, each manager defines its own place. The manager is the center of the place, and the place is the set of all SAMOA clients who are physically proximate to the manager-that is, those devices that are connected to the manager device by a routing path of a maximum length of h network hops, called the place radius. In SAMOA, network hops represent the distance between two physically connected entities. For example, two entities whose devices are within each other's communication range have a distance of one hop. We don't determine the set of clients in a place a priori; rather, it dynamically changes as users move and devices are disconnected and reconnected. Depending on the application deployment scenario, different mappings of the place abstraction are possible,

either fixed or mobile. For example, a place might define the set of users whose devices are currently connected to the same wireless cell or to the same mobile ad hoc network (MANET). Places can overlap, or they can be defined by more than one manager-for example, two managers could be allocated at a one-hop distance. Users can freely roam among places and might be clients of more than one place at any time.

6.3.2 Profiles Model

All SAMOA entities, i.e., places and users, are associated with unique identifiers and profiles describing their characteristics. Profiles have a modular structure comprising different parts, each grouping metadata with a common logical meaning.

Place Profiles

A place profile has two parts:

- The *identification* part includes a unique identifier, a name, and a description of the physical place.

- The *activity* part includes all of the social activities that characterize the place and that all members of that place share. For example, a bookshop's profile might include activities such as shopping and reading.

SAMOA's place profiles account not only for users' social preferences, but also for the relationships between people and the places where they're located and where interactions are likely to occur. The underlying assumption is that the places where users move and operate will influence their activities and interactions with other users.

User Profiles

The profiles for all SAMOA users (managers, clients, and members) include identification and preference parts.

- The *identification* part provides user naming information, such as a personal identifier, and describes user properties, such as age, gender, and education.
- The *preference* part defines user activities. In particular, the activities the user is interested in and, for each of these activities, the user's specific preferences. For example, in the user profile in figure 2a, the user is interested in the shopping activity, and mostly prefers books about history that cost less than 80 euros and are at the superstore Harrod's.

Discovery Profiles

SAMOA managers also have a discovery profile associated with each place they manage. The discovery profile defines the *preferences* clients must match to join the manager's social network. Similarly to the user profile preferences, discovery profile preferences include desired client attributes for each activity. For instance, a manager's discovery profile might state that he or she is looking for other users of the same age who are interested in the shopping activity, preferably in buying books.

6.3.3 Social Network Extraction Model

SAMOA allows managers to exploit two different SNs: a place-dependent and a global SN. The place-dependent SN provides the visibility of only the members currently co-located with the network manager, whereas the global SN persistently records the whole set of place-dependent SNs dynamically created over time as the manager moves across

```
<profile:User rdf:ID="Alice">
 <profile:hasProfile>
   <profile:UserProfile rdf:ID="Alice_User_Profile">

    <profile:has_personal_data>
      <profile:Personal_Data rdf:ID="Alice_Personal_Data">
        <profile:age
            rdf:datatype="http://www.w3.org/2001/XMLSchema#int">30
         </profile:age>
        <profile:name xml:lang="en">Alice Smith</profile:name>
         <profile:Education
               rdf:resource="&edu-ont;ComputerScience_BSc"/>
      </profile:Personal_Data>
    </profile:has_personal_data>

    <profile:has_activity>
      <activities:Shopping rdf:ID="Shopping_Inst_1">
        <profile:activity_preference>
          <activities:Shopping_Preference rdf:ID="Shopping_Pref_1">
            <profile:has_pref_item>
              <shopping:Superstore rdf:ID="Harrods"/>
             </profile:has_pref_item>
             <profile:has_pref_item>
                <shopping:Book rdf:ID="Book_1">
                  <shopping:book_topic
                 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
                         History</shopping:book_topic>
                  <shopping:max_price
                 rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
                         80</shopping:max_price>
                 </shopping:Book>
              </profile:has_pref_item>
            </activities:ShoppingPreference>
          </profile:activity_preference>
       </activities:Shopping>
     </profile:has_activity>
     ...
   </profile:UserProfile>
 </profile:hasProfile>
</profile:User>
```

Id

Activity

Preference

Figure 6.2: SAMOA user profile example.

different places. SAMOA determines place-dependent SNs by adopting semantic-based matching techniques (see Figure 6.3.

In particular, SAMOA provides two matching algorithms. The first algorithm operates on User and Place profiles to identify a first set of eligible members within the discovery scope of a place. By applying the matching algorithm to all eligible members co-located in the place, SAMOA builds the manager's SN for that place. Section 6.5.3 provides the details of SAMOA matching algorithms.

With regard to the global SN of a manager, it is incrementally built by maintaining information about the members of all transient place-dependent SNs. In particular, for any new member in a place-dependent SN her related profile is also permanently included in the manager's global SN. Let us note that the provision of place-dependent SNs allows

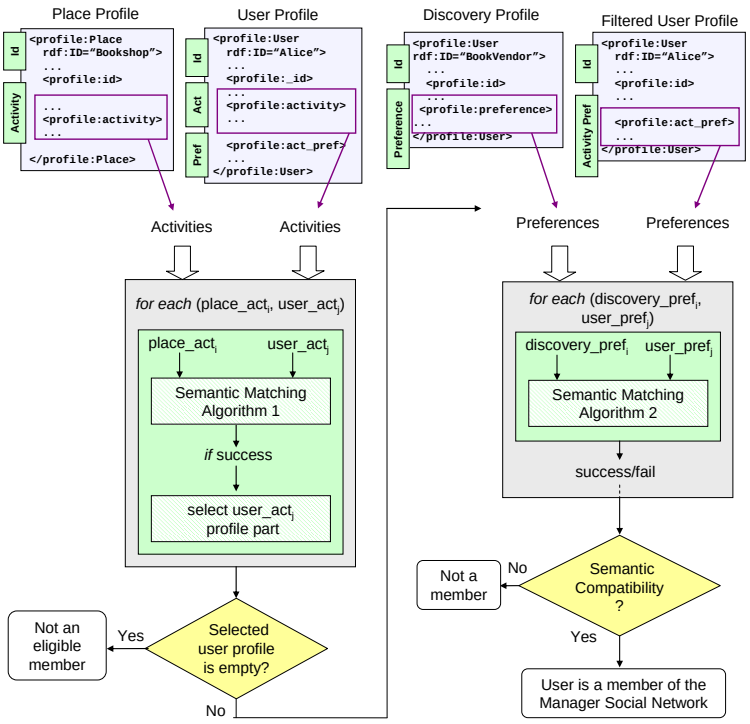

Figure 6.3: SAMOA profile-based social network extraction.

managers to easier discover co-located users of interest when willing to establish with them only one-shot and transient interactions, whereas global SNs provide managers with the possibility to create application-dependent past interaction histories that can enable more complex collaboration strategies and patterns.

6.4 Middleware Architecture

SAMOA middleware has a layered architecture, built on top of the Java Virtual Machine, organized in two logical layers: the Basic Service Layer and Social Network Management Layer (shown in Figure 6.4).

The **Basic Service Layer** provides the needed facilities to support naming, detection of presence of co-located SAMOA entities and communication. It includes the

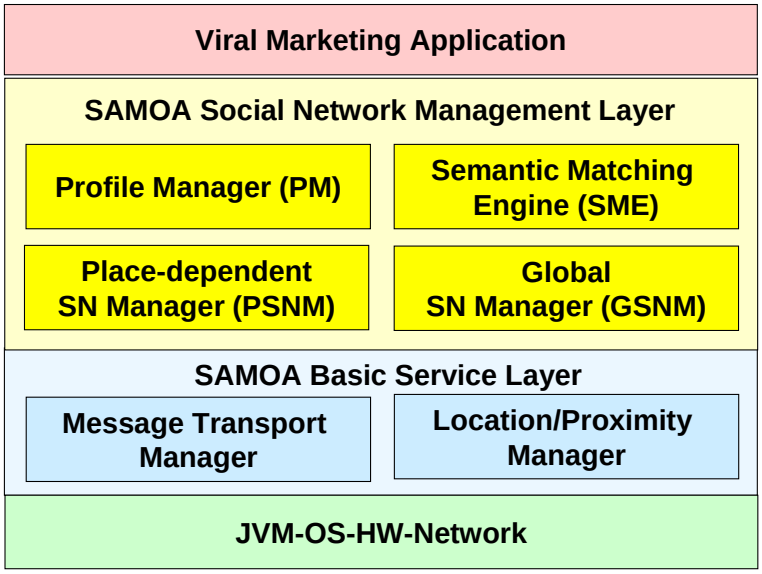

Figure 6.4: SAMOA middleware layered architecture.

Message Transport Manager, which supports communication between connected users, and the **Location/Proximity Manager**, which provides basic naming and monitoring support for currently connected users' availability, and advertisement of co-located peers.

The **Social Network Management Layer** includes facilities for semantic-based SN extraction and management. In particular, the **Profile Manager** provides graphic tools for the specification/checking for correctness and for user/discovery/place profile distribution to interested SAMOA entities and support facilities. The **Semantic Matching Engine** supports semantic matchmaking for social network extraction. The **Place-dependent Social Network Manager** manages the place-dependent SN, by keeping track of all manager SN members which are currently co-located, while the **Global Social Network Manager** creates the global SN by maintaining all place-dependent SNs.

6.5 Prototype Implementation

We have developed a Java-based prototype version of the SAMOA framework to be deployed either in MANET scenarios, or in wireless Internet scenarios, where fixed nodes acting as access points provide connection to the Internet of mobile nodes equipped with IEEE 802.11-compliant wireless connectivity.

6.5.1 Basic Service Layer

The Basic Service Layer has been implemented by specializing the AGAPE middleware architecture [27] In particular, the Message Transport Manager (MTM) implements UDP-based point-to-point and multipoint communication patterns. The MTM point-to-point communication support permits to send messages to a host identified by a known IP. The MTM multipoint communication support permits to broadcast a message to several SAMOA entities allocated in the same place, by following a broadcast protocol for cell-based environments and a flooding protocol for MANET settings.

The Location/Proximity Manager (L/PM) generates and assigns user PIDs and place IDs by exploiting a naming approach that statistically ensures identifier uniqueness. In addition, L/PM permits SAMOA entities to advertise their on-line availability by broadcasting advertisement messages at regular times. Advertisement messages include the PID/IDs of the entity, and its IP address. All SAMOA entities rely on L/PM to sense incoming advertisements and to build a table of currently sensed co-located users from received advertisements. Table entries are associated with timestamps: if L/PM does not receive beacons from an entity within a defined threshold, the associated entry is removed and the entity device is considered disconnected. L/PM disseminates advertisements only within the scope of the place by coordinating with the MPM. The physical area delimiting a place is determined once, at place deployment time.

6.5.2 Social Network Management Layer

The Profile Manager (PM) provides graphic tools for the specification/checking for correctness and for user/discovery/place profile distribution to interested SAMOA entities and support facilities. As Figure 6.2 shows, SAMOA adopts OWL-based formats for profile representation. The Java-based ontology editor Protg enables profile visualization and browsing.

The Semantic Matching Engine (SME) supports semantic matchmaking according to the two algorithms described in Section 3.2. In the current implementation, SME relies on the subsumption reasoning capabilities of the Pellet reasoner [8], while OWL ontologies are stored and accessed via the semantic web framework Jena [13]. In particular, the Pellet reasoner is accessed via Jena APIs and SPARQL queries [1].

The Place-dependent Social Network Manager (PSNM) manages the place-dependent SN, i.e., creates and maintains a table including all manager SN members which are currently co-located. PSNM coordinates with PM to obtain the PID and UP part of members. When a member connects/disconnects from the place PSNM updates the table to reflect the change (via coordination with L/PM). Each entry in the place-dependent SN includes several information about members such as, PID, device IP, along with the returned UP part semantically compatible with the place profile.

The Global Social Network Manager (GSNM) creates the global SN by maintaining and storing all place-dependent SNs in a dedicated table. Each table entry stores all PIDs, UPs of all members belonging to the manager's SN. In addition, for each member the place PP and manager DP that guided member selection are stored.

6.5.3 Social Matchmaking Algorithms

SAMOA provides two matching algorithms. The first algorithm operates on User and Place profiles to identify a first set of eligible members within the discovery scope of a place. The algorithm compares all UPs of the users currently located in the manager's place with the place's PP. Only those users whose UPs have activities with semantic relations with the PP's activities become eligible members. The algorithm returns, if any, the semantically compatible UP parts of eligible members, as shown in Figure FIGURA A. The second matching algorithm is used to elect as members only those users whose attributes semantically match with the preferences included in the DP of the place's manager. In particular, the algorithm iteratively analyzes all UP parts returned by the first algorithm to determine whether the preferences in the UPs of eligible members semantically match with the preferences in the manager DP (see Figure 6.3). By applying the matching algorithm to all eligible members co-located in the place, SAMOA builds the manager's SN for that place. Figure 6.5 shows the details of SAMOA matching algorithms.

Both matching algorithms exploit Description Logic-based subsumption reasoning to determine whether a particular individual is an instance of a certain class. Toward this goal place activities and preferences in the manager's DP are represented as classes, while user activities and preferences in the UP of eligible members are defined as instances. In addition, activity/preference classes are defined by constraining their specific properties to assume a certain (range of) values, e.g., a preference class about shopping in the manager DP might be defined by constraining the property representing the purchased object to assume the value "book". Let us suppose that a user preference instance has the property representing the purchased object set to the "book about history" value. In this case, the matching algorithm infers that the user preference is an instance of the preference class in the manager DP.

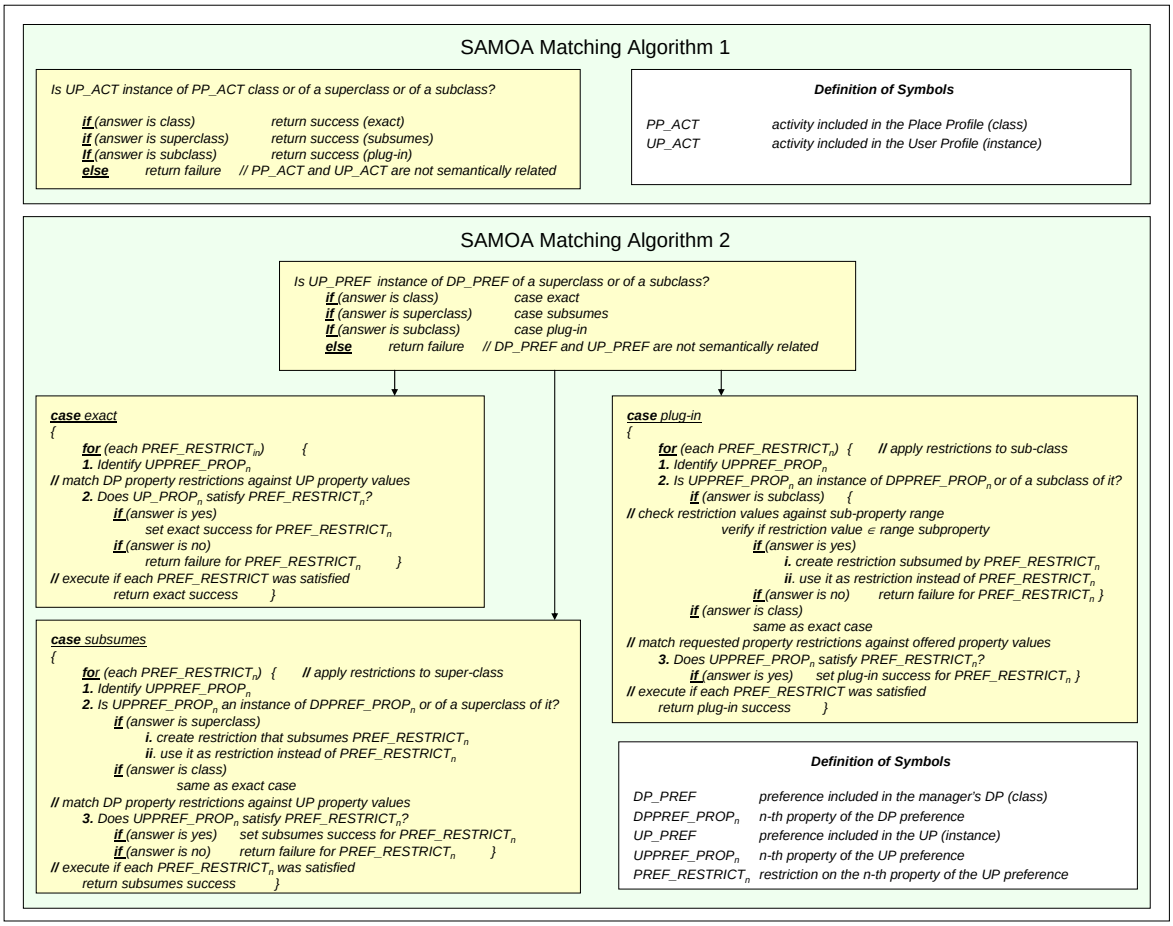

Figure 6.5: SAMOA semantic matching algorithms.

Let us also note that the matching algorithms are able to recognize different semantic relationships on the basis of the subclass relationships defined in the activity/preference ontologies. In particular, both algorithms recognize three semantic similarity relationships. The user activity/preference may be an instance of the activity/preference class in the manager PP/DP (exact case), or an instance of a more generic activity/preference class (subsumes case), or an instance of a more specialized activity/preference class (plug-in case).

6.6 Case Study

This section provides some functioning insights and performance evaluations about our SAMOA framework in a viral deployment marketing scenario. In particular, here we describe how SAMOA support facilities allow vendors/customers to build SNs to distribute specific product promotion advertisements.

SAMOA support permits vendors to forward promotional messages, e.g., book discounts, to customers currently located in their bookshops (on the basis of the vendor's place-dependent SN) and to all customers that previously visited their bookshops (on the basis of the vendor's global SN). Customers can contribute to promotional information spreading, too. Once a customer receives a promotion advertisement he can exploit SAMOA to forward the information to all nearby customers (on the basis of the customer's place-dependent SN) and to all customers previously encountered in all visited bookshops (on the basis of the customer's global SN). In particular, information forwarding follows a word-of-mouth model based on the impromptu customer's encounters during shopping activities.

6.6.1 Application Deployment

As test-bed scenario for our viral-marketing application prototype developed on top of SAMOA we have considered the case of a shopping mall hosting various shops including one bookshop. Connectivity is provided by IEEE 802.11-compliant access points (APs). In particular, each shop has one AP that provides wireless connectivity to all customers within the AP coverage area. Customers are equipped with laptops with IEEE 802.11b/g wireless cards running the viral-marketing application. The bookshop has one server installed hosting the viral-marketing application prototype. The server host and customer devices run all SAMOA support facilities. In our deployment scenario the book-vendor and the customers play all the manager role: they define their places and create their own SNs.

Let us note that the place defined by the vendor is fixed with the server acting as the place center, whereas the customer's place is mobile (being the customer roaming) and its allocation is determined by the network cell of the shop the customer is currently attached to, e.g., the bookshop cell. Given this deployment setting, the discovery scope for SN extraction is restricted to only SAMOA entities whose devices are connected to the same wireless cell. It is worth noticing that vendors and customers appear as clients when in places managed by others.

The realization of the SAMOA-based viral-marketing application consists of two distinct phases.

- In the *application programming* phase application developers define and code only the application functionalities without dealing with SN management issues.
- In the *application deployment* phase developers specify application-specific configuration parameters and the SAMOA profiles needed to guide appropriate SN extraction, transparently to the application.

In particular, at application deployment time, the viral-marketing application running on the server managed by the book vendor allows the vendor to describe the commercial promotions for potentially interested people, the bookshop Place Profile, as well as the Discovery Profile and the vendor's User Profile. Our prototype promotion descriptions include various information, such as shop contacting information (name, address, telephone number, and/or e-mail), or the set of books which are sold at a discounted price with their prices. Figure 6.2 and 6.6 depict some profile examples valid for the application prototype, e.g., a customer's UP, the bookshop's PP describing the activities characterizing the bookshop, e.g., shopping and reading, and the vendor's DP defining the manager's preference for customers interested only in buying books. Similar considerations apply to

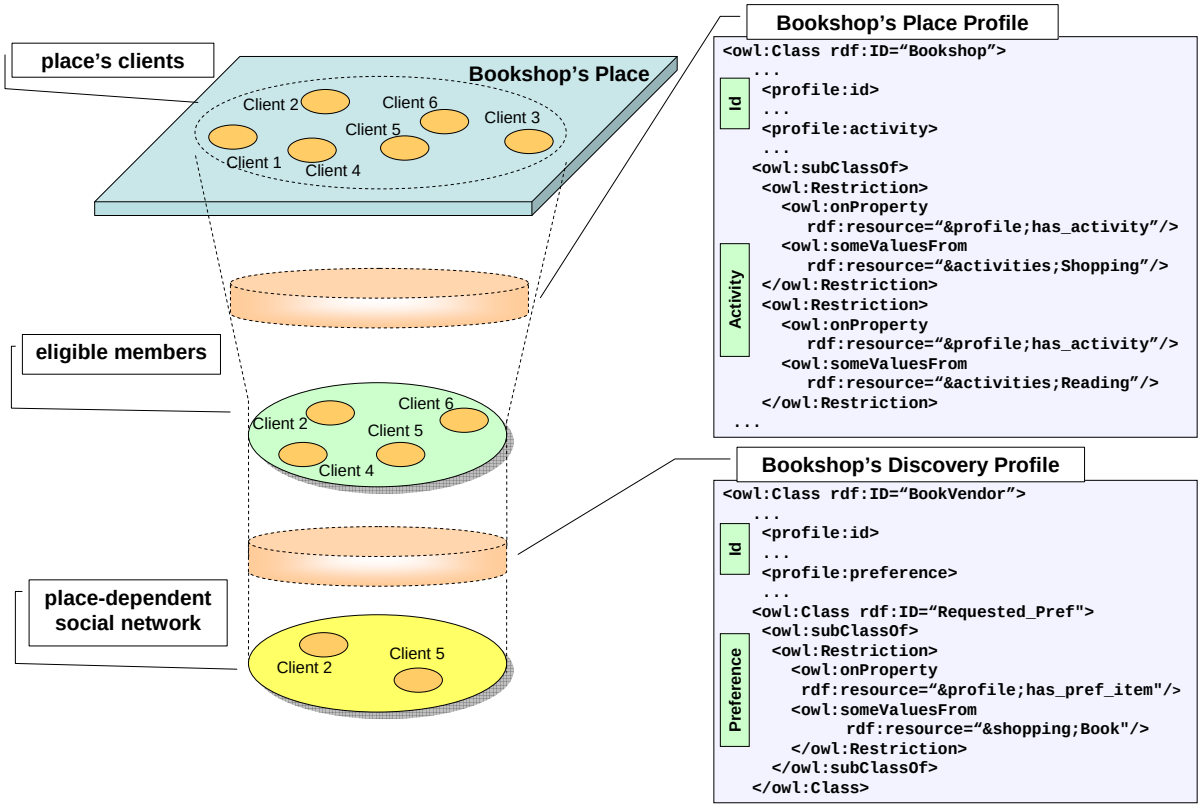

Figure 6.6: Bookshop's UP and DP and their use in social network extraction.

the viral-marketing applications running on the customer's devices.

6.6.2 Social Network Extraction

In the following, we illustrate how SAMOA allows the vendor to extract his SN by focusing on the main steps. Similar considerations apply also to the customer's SN extraction. To determine his place-dependent SN, the vendor has to first identify the place. In our considered test-bed setting, the server installed acts as the manager and his managed place is mapped onto the bookshop wireless cell. Figure 6.7 shows the interaction flow among SAMOA support facilities to build the vendor's place dependent SN. When a customer enters the bookshop, her device connects to the locally available wireless cell. The L/PM

instance running on her device advertises her availability and also detects the availability of a new place, i.e., the vendor's place. Then, the PM instance running on the customer's device coordinates with the PM component running on vendor's device to first obtain the PP. Upon profile reception, the PM instance running on customer's device coordinates with the SME facility to filter the customer's UP according to the bookshop's PP: in this case, only customer preferences related to the activities of shopping and reading are provided to the vendor. If semantic matches are found, the customer is considered an eligible member and the customer's PM coordinates with the vendor's PM to send it only the semantically compatible customer's UP part. When the vendor receives the customer's UP, the PM in the server host coordinates with the local SME facility to verify whether the customer's UP is also semantically compatible with the vendor's DP. In case of successful matching, PM coordinates with both PSNM to include the customer into the vendor's place-dependent SN and GSNM to store the retrieved information in the vendor's global SN table. Let us note that SAMOA SN extraction requires PPs and DPs to be maintained distinct and to be analyzed in separate phases (see Figure 6.6). This has several benefits. The manager communicates to co-located customers only the PP, thus preserving the privacy of his specific DP containing possibly confidential marketing strategy choices. Similarly, customers distribute their UPs only to vendors providing places with activities of interest. In addition, the distinction between PPs and DPs allows to distribute the SN extraction overhead among all users: PP semantic analysis is performed only on customer devices, whereas semantic matching between DP and UP is carried out only on vendor devices.

As a final remark, let us note the different use of the vendor's place-dependent and global SNs. The application module running on the server host exploits the vendor's place-dependent SN whenever a new customer enters the bookshop to send him a promotion. On the other hand, the visibility of the vendor's global SN permits the vendor to optimize

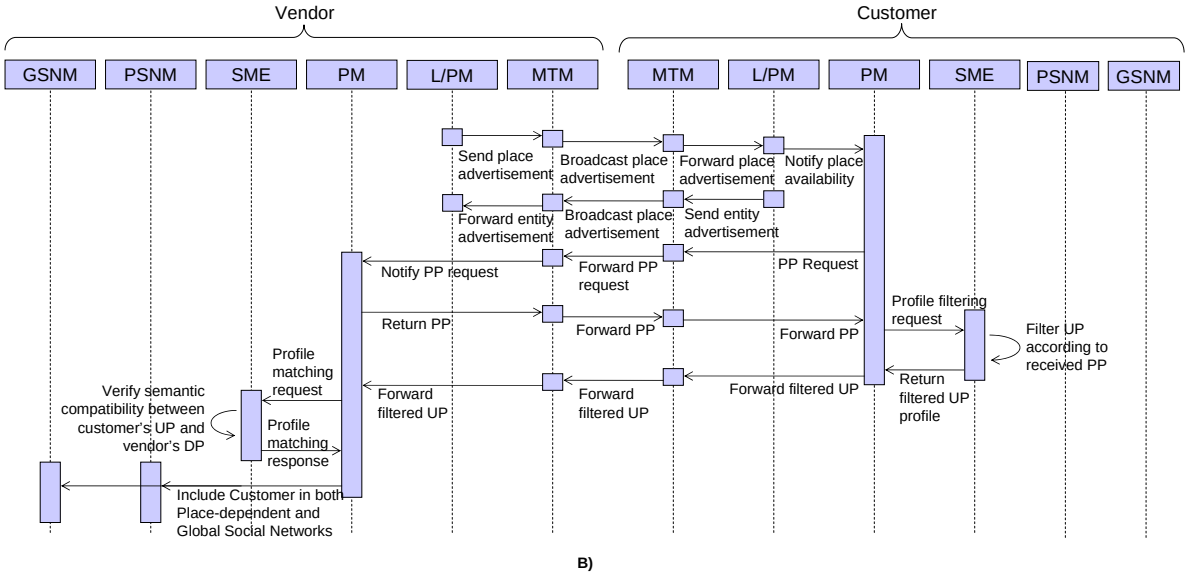

Figure 6.7: Interaction flow diagrams in the case study.

marketing strategies by browsing information about all customers that have visited the bookshop over time. For example, if the vendor detects that most of her customers study computer science engineering she may decide to tailor her promotion accordingly.

6.7 Evaluation

The exploitation of a semantic middleware to support SN extraction, such as SAMOA, introduces different forms of overhead, depending on both the deployment environment and the performance of the different middleware facilities. For the sake of briefness, we report here some evaluations about the quality of SAMOA matching algorithms and the overhead introduced by the adoption of semantic metadata/techniques.

To evaluate the quality of the matching algorithms, we have considered a test-bed activity and preference ontology modeled as a hierarchical classification tree, whose depth (maximum degree of activity/preference specialization) is 4 and breadth (multiplicity of

User	Place	UP Preferences	PP Activities	DP Preferences	Time (ms)
Geek Bookshop	2	2	4	2	3484
Master Yoda	3	3	-	-	4363
Luke Skywalker	1	1	-	-	874
The Bluesman's Pub	1	1	3	1	4835
Alice Smith	2	2	3	3	2667

Table 6.1: Semantic model instantiation time.

activity/preference related concepts) is 3. In our tests we have considered user/ place profiles with a variable number of activities, from 1 to 4, and place/ discovery profiles either none or one preference per activity. To evaluate the quality of our matching algorithms, we have measured recall, i.e., the extent to which all socially compatible users are included in the network (by avoiding false negatives), and precision, i.e., the extent to which only socially compatible users are included in the network (by avoiding false positives) [25]. Being our matching algorithm complete, its recall is optimal. SAMOA has also demonstrated a good level of precision, mostly thanks to its ability of looking for the manager-specified preference values only in the semantically correct activity type, thus reducing false positives.

In addition, we have evaluated the impact of semantic-based matching on SN extraction time. Our tests have been performed on an Intel Core 2 Duo processor, equipped with 1Gb of RAM, running Windows XP Home Edition. The prototype uses Pellet 1.5.1 and Jena 2.5.4, on JRE 1.6.0. Table 6.1 shows the average times needed to load and instantiate Jena semantic models of the UP, PP and DP (when avilable) with different numbers of activities and preferences. Instantiation time keeps below 5 seconds, which represents a reasonable setup time for an application like SAMOA. Table 6.2 reports the average times for both semantic matching algorithms execution. This test also shows that semantic-based

User	Place	Time (ms)	SN inclusion
Jack Bauer	The Bluesman's Pub	2995	refused
Jack Bauer	Geek Bookshop	79	refused
Jack Bauer	Alice Smith	2854	included
John Locke	The Bluesman's Pub	62	refused
John Locke	Geek Bookshop	3394	included
John Locke	Alice Smith	53	refused
William Adama	The Bluesman's Pub	64	refused
William Adama	Geek Bookshop	712	refused
William Adama	Alice Smith	3441	refused
Alice Smith	The Bluesman's Pub	2604	included
Alice Smith	Geek Bookshop	21081	included

Table 6.2: Total execution time for semantic social matchmaking.

social network creation time, which is clearly below 4 seconds in the worst case, is compatible with the targeted application. Let us note that, in case the user profile does not include any activity that is semantically connected to the discovery profile of the considered place, the response time drops down to less than 1 second. This performance improvement is due to an optimized implementation of the first algorithm: before iteratively comparing each activity in the UP and DP, SME performs a preliminary reasoning to check whether UP actually contains DP-semantically compatible activities. Although this solution introduces an additional reasoning step, the obtained pruning allows to discard all incompatible UPs, thus significantly reducing the total reasoning time, as shown in Table 6.2. It is also worth noting that SN creation time also depends on the communication overhead needed to transfer profiles over the wireless connection. However, the overhead is heavily influenced by the quality/throughput of the underlying wireless connection, independently from SAMOA

functioning. For this reason we have not focused on the evaluation of this overhead.

6.8 Related Work

Over the past few years social computing has been the subject of active research and development efforts in both academia and industry. Several social software systems are available, targeting different application domains and exhibiting different technical features, ranging from the information sources considered for social relation extraction to the criteria and techniques adopted for SN building. The aim of this section is not to provide an exhaustive survey of SN systems, but to outline the key properties of most widespread solutions and to identify emerging requirements that have not been fully addressed yet by existing research on social computing.

From an historical prospective, Internet has given a great impulse to the research in the social computing field. Several Internet-based prototypes have emerged that address various application domains, including the automatic identification of commonly acknowledged experts in a field, the study of social dynamics characterizing a certain social system , the improvement of search engine ranking algorithms, just to mention a few [68]. The shared underlying assumption is to consider the Internet as the main source of empirical data from which to extract social relations and patterns. Web-pages, scientific publication data-bases, XML-based FOAF files [2], personal or organization mail repositories, usenet posts represent examples of different commonly used information sources. Current Internet-based social systems adopt different social network analysis approaches, mainly based on data-mining techniques: co-occurrence of names in a web-page, co-citation/co-authoring of a scientific paper, identities of the sender and the recipient of a mail can provide a suitable base to identify socially-bound individuals [81, 68, 93]. Only recently few other approaches

based on ontologies have been proposed to model and infer relevant social relationships among individuals [74].

The new emerging trend in the social computing field is represented by the design of SN systems for ubiquitous computing environments. However, ubiquitous computing environments are far more dynamic and heterogeneous than Internet-based deployment scenarios, thus calling for novel solutions that should follow different design guidelines from their Internet-based counterparts. The set of potentially available SN members continuously varies due to user mobility and cannot be statically pre-determined. Interactions among individuals are typically opportunistic, transient and serendipitous and are more likely to involve users in physical proximity. In addition, the large-scale and dynamicity of ubiquitous environments make SN management challenging. It is, therefore, crucial to identify appropriate criteria for delimiting the searching space for SN members (discovery scope): social interests and affinities are too coarse-grained parameters, whereas the physical place where users are likely to establish interactions has been recognized as an additional first-level criteria for SN extraction [46, 58]. The recognition of the need for a shift to a location-centric social computing paradigm has guided the design and development of recent social networking solutions. Systems like LoveGety, ProxyLady, SocialNet, BEDD, Ulocate, ActiveBadge, Meme Tags and Serendipity exploit proximity or co-location visibility to verify whether co-located users have affinities, by inferring relations either by discovering patterns of co-location or by matching profiles among people who are close to each other [25].

Current social networking solutions represent interesting steps forward, but are still more proof-of-concept application prototypes of single management aspects rather than comprehensive frameworks for supporting the design, development and deployment of anytime and anywhere social networking services. To the best of our knowledge, all literature

proposals are built on top of the network layer and tend to provide a dedicated support for specific applications, directly embedded into the application. This approach has some limitations. First, ad-hoc supports can be hardly reutilized in different application domains, thus requiring to build from scratch a new support system anytime a new application is developed. In addition, building social networking applications on top of the network layer might be extremely tedious and error-prone as it requires developers to explicitly deal with all issues related to user/device mobility, intermittent connectivity and availability of SN members. On the contrary, SAMOA provides suitable support for addressing SN management details, such as user location detection/tracking, user profiling, and social matchmaking. Being a middleware level solution, it allows application developers to focus only on the design and development of the application logic while being relieved from the burden of addressing low-level SN management details. This significantly simplifies and accelerates application development. In addition, SAMOA middleware can be used in different social computing applications, thus favoring the interoperation between applications and the rapid prototyping of social computing applications

In addition, differently from most SN management solutions, SAMOA adopts semantic representation models and languages to describe user context at the proper level of abstraction, while enabling automated reasoning on context representations. This allows the greater interoperability even in an open and dynamic deployment scenario, such as the ubiquitous one, where it is not possible to make a-priori assumptions about the way user contexts are described.

6.9 Ongoing Work

The design and development of SN services for ubiquitous environments is a challenging task. We have implemented the SAMOA framework by following two main design guidelines, namely context-awareness in SN extraction and the adoption of semantic technologies to perform social matchmaking of SN members. The encouraging results coming from our early experiences with SAMOA are stimulating further research to improve the framework design. We are working along several directions, primarily on the SAMOA integration with security supports for addressing privacy issues, crucial to leverage the SAMOA adoption in untrusted ubiquitous environments. In addition, we are working to evaluate SAMOA applicability and usability in several other application prototypes on top of it, such as an e-campus application supporting the creation of social networks inside a faculty campus. We are also planning to further develop the SAMOA framework to provide support for social applications, such as recommender systems.

6.10 Chapter Summary

This chapter has presented our middleware SAMOA, which exploits semantic techniques to perform context-aware social network extraction based on user activities and current location. To outline the requirements of SN management in pervasive environments, the chapter has discussed a viral marketing application scenario. It has shown SAMOA SN management and extraction model based on semantic metadata, and SAMOA semantic algorithms to match user request against service offer based on user and place activities/ preferences. It has also provided details about a viral marketing prototype implementation built on top of SAMOA and evaluated the performance of the prototype. After discussing relevant related work, the chapter has given insights on current research on SAMOA.

Chapter 7

Conclusions

A little semantics goes a long way.
James Hendler

This chapter is to summarize our research and present the contributions of this dissertation. We also discuss the results of our work by describing lessons learnt and issues that still need to be addressed. Finally, we outline possible future research directions.

7.1 Thesis Summary

The dynamicity and heterogeneity that characterize pervasive environments raise new challenges in the design of mobile middleware. Novel middleware solutions should support mobile computing applications by adapting their behavior to the frequent changes in execution context, i.e., middleware should become context-aware. The adoption of metadata represents a promising approach to the design of context-aware middleware solutions. In most current middleware solutions, however, the meaning of used metadata is only known to developers and/or system administrators. We believe that the inability of the middleware

platform to automatically acquire and process knowledge about the underlying system has hindered until now the full achievement of context-awareness in pervasive applications. Thus we claim that a novel generation of middleware should support context-awareness by exploiting semantic metadata, whose meaning is explicitly defined in a machine-understandable form and can thus be acquired and processed by software applications.

According to this claim, this thesis has fully investigated novel design guidelines and implementation options for semantic-enabled middleware solutions targeted to pervasive environments.

7.2 Thesis Contributions

Compared to existing solutions to support context-aware pervasive applications, our middleware-level approach based on semantic metadata is novel in that provides a combination of advantages. First, it supports application developers with a set of abstractions and mechanisms to propagate context visibility up to the application level. Therefore, developers can focus only on the application logic, which is not altered by context-aware adaptation strategies enforced by the middleware layer. In addition, we have shown that single middleware components can be reutilized in different applications by customizing metadata to describe the application-specific knowledge domain. Moreover, the exploitation of semantic metadata to represent and reason about context allows to infer context information from available data, thus empowering the middleware to make appropriate/complex management choices based on changing context conditions. Semantic metadata also favor dynamic interoperability between entities sharing little or no prior mutual knowledge by allowing the exchange of *meaning* in a machine-processable form. Finally, our metadata-based middleware approach can be applied to address the issue of providing resource-constrained mobile

devices with proper semantic support.

In a more organized view, this thesis provided several contributions to research in the field of mobile middleware:

1. *The definition of a metadata model to represent and reason about context.*

   Among the different possible types of metadata, in this thesis we consider profiles and policies. Profiles represent characteristics, capabilities and requirements of system components, such as users, devices and services. Policies express the choices ruling system behavior, in terms of the actions subjects can/must operate upon resources. We adopt semantic languages and technologies to specify and manage metadata. Semantic languages permit the explicit representation at a high level of abstraction of interacting entities, e.g., services, resources and users, and their context, e.g., current location of users/devices, state of resources, user preferences, and device characteristics, while enabling automated reasoning about this representation. This favors the dynamic interoperability and mutual comprehension between entities sharing little or no prior knowledge about each other. In addition, the adoption of semantic languages for metadata specification simplifies metadata reuse and extensibility, and facilitates the analysis of potential conflicts and inconsistencies. In particular, we express semantic metadata using a Semantic Web standard language, i.e., the Web Ontology Language (OWL) [20].

2. *The definition of a model for the design and development of context-aware middleware based on semantic metadata.*

   Context-aware middleware should be able to collect, represent and reason about the context, and to propagate this information up to the application level. In this thesis we address this requirement by cleanly separating context-dependent application

management from application logic. This separation of concerns is crucial to reduce the complexity of developing applications for pervasive environments and to favor rapid application prototyping, runtime configuration, and maintenance. Context information and context-aware adaptation strategies are expressed at a high level of abstraction by means of semantic metadata, and exploited to take context-aware management decisions that do not impact the application logic.

3. *The design of three novel middleware architectures and the development of a prototypal implementation for each of these architectures.*

   These prototypes provide an implementation of the metadata-based model for context-aware middleware, offer a wide range of mechanisms to collect and manage relevant context information, and propagate it up to the application level. A key feature common to the developed middleware infrastructures is the exploitation of semantic technologies to represent and reason about context information. In particular, the developed middleware architectures are targeted at different pervasive computing scenarios that might benefit from the enhancement of existing solutions with context-awareness, namely: (i) the MIDAS framework supports personalized service discovery for mobile users; (ii) the Proteus framework performs context-aware access control on resources hosted on portable devices; (iii) the SAMOA framework provides support for the creation and management of context-aware social networks.

4. *The proposal of a viable approach to portability issues raised by the adoption of semantic support services in pervasive applications.*

   To address the issue of providing resource-constrained portable devices with adequate context-aware middleware support, we have applied our metadata model to describe semantic support services, mobile device properties, as well as configuration strate-

gies needed to deploy semantic support components on mobile devices. This allows to exploit the same management and adaptation mechanisms developed for context-awareness to properly configure semantic support based on mobile device properties. In particular, we have extended our discovery middleware MIDAS to support the dynamically configurable deployment of semantic support services on-board of mobile devices, and we have provided a prototype implementation of this extension.

7.3 Discussion

The design of context-aware middleware for pervasive applications is a challenging task. This section discusses open issues and lessons learned from our experiences of developing context-aware middleware support solutions.

7.3.1 Lessons Learned

- *Managing context information is difficult.*

  Most existing work in the area of context management has been focusing on representing context information at a right level of abstraction and granularity. Since context modeling can be thought as a specific application of knowledge representation (KR), previously defined KR models and techniques have been applied to address the particular issue of representing context. In addition, semantic approaches, which exploit ontologies, recognize and try to address the need of interoperability between different context representation models and languages. Despite these significant initial efforts, however, several issues remain open with regard to context information acquisition and processing. In particular, appropriate models and mechanisms are needed to aggregate simple/raw context data into higher level information that can be exploited at the application level. Being this aggregation process strongly application-dependent,

most solutions have been developed for specific purposes, thus lacking support for reusability in different application scenarios. More generally, we believe that research on context-aware middleware still needs commonly agreed design guidelines according to which context should be modeled, represented, processed and presented to the application layer. Another crucial issue that has been only marginally addressed concerns the quality of context information (QoC), in terms of sensor reliability, updating mechanisms, correctness of the aggregation process and consistency of context information obtained from disparate sources. Since application behavior should be adapted based on context information, it is fundamental for context-aware middleware to take into account QoC parameters to appropriately take and possibly adjust system management decisions.

- *A systematic approach to the provisioning of semantic support to mobile devices with different capabilities is still to be defined.*

  Despite various solutions for specific applications have been proposed and prototypically implemented, to the best of our knowledge there is neither a generic framework nor an agreed methodology for semantic support configuration. Some preliminary studies have been conducted to evaluate the efficiency and portability of semantic support services, like for example performance measurements on loading and reasoning over huge ontology bases [104]. Research in the area of semantic support is very active, new software tools are developed and existing ones are continuously improved, such as inference engines or ontology base management systems [8, 13]. Given the ongoing technical evolution of available semantic support components, at present it might be difficult to make a reliable assessment about semantic support portability. Moreover, it is worth noting that the issue of providing portable devices with semantic support can be often resolved to a configuration engineering problem. This allows to

exploit well-established techniques and solutions that have successfully been applied to address similar issues of distributed computing applications.

- *Semantic technologies can make system management and control very complex.*
  The enhancement of middleware solutions with context-awareness relieves mobile users from the burden of manually adapting their applications to changing context, but tends in turn to increase the complexity of management tasks demanded to the middleware layer. Semantic techniques promise to simplify context-aware system management by providing support to the middleware for context representation and automated reasoning. In this respect we might say that the adoption of semantic techniques moves system complexity and related issues one step backward, i.e., from middleware management support to knowledge representation and processing models. In fact appropriate context-aware adaptation can be achieved as long as semantic metadata describing interacting entities and their context are adequately represented and processed. In other words, semantic technologies must be *reliable* in order to have context-aware middleware solutions built on them. This means, for example, that a reasoning process will terminate in reasonable time, but also that an ontology base is semantically consistent in that it does not contain incorrect knowledge about the system. The need to ensure such semantic reliability makes the design of context metadata and context processing tools a challenging task. From our experience we suggest that a viable tradeoff between metadata expressivity and issues deriving from their management might be reached by limiting the complexity of adopted semantic models.

7.3.2 Open Issues

- *Metadata currently lack specification guidelines and standard models.*

  Several standardization efforts have been undertaken in recent years to reach common agreements on semantic knowledge representation frameworks and languages. Due to the W3C Semantic Web activity, the so-called Semantic Web layer cake has been added several layers that generally involve large consensus both from academia and industry. As far as metadata specification is concerned, some important standardization efforts have produced relevant results, especially regarding service profiles (with the WSDL specification, for example) and device profiles (with the CC/PP W3C standard). However, a recognized standard for user profiles is still lacking, and profiles are defined based on specific application requirements. Policy specification is even more problematic since several different approaches to policies have been proposed in research literature, from rule-based to ontology-based models, to business-oriented and service level agreement models. Although policy-based management applies to a set of disparate application areas that could hardly be harmonized and/or classified under common criteria, nonetheless we believe that defining common design guidelines for semantic policy specification would represent a crucial step towards the wide adoption of semantic metadata.

- *Design guidelines for policy-based management of context-aware systems are needed.*

  Policies have been extensively applied to network administration tasks, such as configuration, security, recovery, or quality of service. In recent years policies have emerged as a suitable means to control the behavior of complex systems, such as multi-agent systems, and more generally to automate system management. The adoption of a policy based-approach for controlling a system requires an appropriate policy representa-

tion and the design and development of a policy management framework. Similarly to the case of policy specification, design guidelines are reference implementation models are needed to develop policy-based frameworks that are able to manage context-aware systems. In particular, the design of a common framework for policy-based context-aware management should define which operations should be controlled by policies, at which level of granularity, as well as the types of policies to be specified. Despite some initial efforts are starting to emerge, particularly in the field of security, we believe this remains an open issue within current research on context-aware computing.

7.4 Future Research Directions

In this section we outline some research areas that represent possible future directions for our work.

- *Usability of semantic policies.*

  In the new pervasive scenario, unlike traditional computing environments where users of security technologies could be divided into different categories - namely developers of the security system, administrators of security policies and end-users - mobile users should play the role of system administrator of their devices by managing their own security, rather than relying on external security management services. A primary requirement for mobile security is therefore the design of usable and useful security supports that relieve mobile end-users from management decisions, such as how to properly express and translate security policies into device-understandable rules. However, a major obstacle to the adoption of user-defined policies is users' inability to understand and define their own access control policies. To address this issue, we plan to design a novel access control policy model whose main guideline is usability by non

technical end-users, and to develop a policy framework to be deployed on portable devices, such as smart phones.

- *Integration of trust and security management within our middleware frameworks.*
  Given the intrinsic openness of pervasive environments, mobile middleware solutions should be developed by taking security into account as a first-class design principle. Therefore, a possible future research direction includes the enhancement of our MIDAS and SAMOA middleware architectures with adequate support for security and trust management. For example, we plan to develop a secure service discovery framework that performs access control verification not only at service access time, but also during the discovery process. To the best of our knowledge, the idea of controlling access to discovery has been only marginally addressed by research at the state of the art.

- *Adaptation models and mechanisms in context-aware middleware.* The middleware ability to support application adaptation remains a crucial issue for the development of context-aware middleware solutions. Beside the design of simplified metadata model clearly understandable by non technical users, powerful and manageable adaptation mechanisms are still needed to perform actual context-aware application management. Metadata themselves might be adapted to changing context. We have started working along this research direction within the Proteus framework and plan to explore it further.

- *Personalized service composition of pervasive services.*
  Mobile users should be supported not only in searching and retrieving services of interest, but also in composing them to obtain more complex and/or higher level functionalities. Analogously to service discovery, personalization of service composi-

tion via context-awareness represents a key feature to enable pervasive service provisioning scenarios. Our middleware architecture for service discovery MIDAS might be extended and enhanced to provide mobile users with support for (semi)automatic composition of pervasive services based on user context.

- *Quality of context-aware middleware solutions.* Current middleware solutions are generally built on the implicit assumption that needed context information is always available, and always reliable. However, this assumption cannot hold in real world scenarios, where context information gathered from the user environment might be often uncertain and/or partly unavailable. This means that middleware based on context metadata should be designed not only to allow context-aware application adaptation, but also to enable adaptation based on variations in the quality of exploited context. We are starting to investigate this research topic, particularly within the Proteus framework, where QoC variations might result in variable risk levels associated to each access control.

Bibliography

[1]

[2] Friend of a friend project. http://xmlns.com/foaf/0.1/.

[3] Jini Technology. http://www.jini.org/.

[4] JSR 179: Location API for J2ME. http://jcp.org/en/jsr/detail?id=179.

[5] Jxta community. https://jxta.dev.java.net/.

[6] Online Community for the Universal Description, Discovery and Integration OASIS Standard. http://uddi.xml.org/.

[7] Overview of SGML Resources. http://www.w3.org/MarkUp/SGML/.

[8] Pellet: The open source OWL DL reasoner. http://pellet.owldl.com/.

[9] Service Location Protocol. Internet Engineering Task Force Request For Comments 2608.

[10] *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), 4-6 June 2003, Lake Como, Italy*. IEEE Computer Society, 2003.

[11] *2nd Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2005), 17-21 July 2005, San Diego, CA, USA*. IEEE Computer Society, 2005.

[12] UAProf Profile Repository, 2008. http://w3development.de/rdf/uaprof_repository/.

[13] The jena semantic web framework, last visited: March 2008. jena.sourceforge.net/.

[14] Alessandra Agostini, Claudio Bettini, and Daniele Riboni. Loosely coupling ontological reasoning with an efficient middleware for context-awareness. In *MobiQuitous* [11], pages 175–182.

[15] Alessandra Agostini, Claudio Bettini, and Daniele Riboni. Experience report: Ontological reasoning for context-aware internet services. In *PerCom Workshops*, pages 8–12. IEEE Computer Society, 2006.

[16] Alessandra Agostini, Claudio Bettini, and Daniele Riboni. *Integrated Profiling of Users, Terminals and Provisioning Environments*, chapter 34, pages 901–937. In [21], 2006.

[17] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[18] Wolf-Tilo Balke and Matthias Wagner. Towards personalized selection of web services. In *WWW (Alternate Paper Tracks)*, 2003.

[19] Jakob E. Bardram. The Java Context Awareness Framework (JCAF) - a service infrastructure and programming framework for context-aware applications. In Hans-Werner Gellersen, Roy Want, and Albrecht Schmidt, editors, *Pervasive*, volume 3468 of *Lecture Notes in Computer Science*, pages 98–115. Springer, 2005.

[20] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL Web Ontology Language Reference, 2004. http://www.w3.org/TR/owl-ref/.

[21] Paolo Bellavista and Antonio Corradi. *The Handbook of Mobile Middleware*. Auerbach, 2006.

[22] Paolo Bellavista, Antonio Corradi, Rebecca Montanari, and Cesare Stefanelli. Context-aware middleware for resource management in the wireless internet. *IEEE Trans. Software Eng.*, 29(12):1086–1099, 2003.

[23] Paolo Bellavista, Antonio Corradi, Rebecca Montanari, and Alessandra Toninelli. Context-aware semantic discovery for next generation mobile systems. *Communications Magazine, Special Issue on Advances in Service Platform Technologies*, 44(9):62–71, 2006.

[24] Martin Berchtold, Christian Decker, Till Riedel, Tobias Zimmer, and Michael Beigl. Using a context quality measure for improving smart appliances. In *ICDCS Workshops*, page 52. IEEE Computer Society, 2007.

[25] Abraham Bernstein and Mark Klein. Towards high-precision service retrieval. In Horrocks and Hendler [57], pages 84–101.

[26] Gregory Biegel, Vinny Cahill, and Mads Haahr. A dynamic proxy based architecture to support distributed java objects in a mobile environment. In *Proceedings of the Confederated International Conferences DOA, CoopIS and ODBASE 2002*, volume 2519 of *Lecture Notes in Computer Science*, pages 809–826. Springer, 2002.

[27] Dario Bottazzi, Antonio Corradi, and Rebecca Montanari. Context-awareness for impromptu collaboration in manets. In *WONS*, pages 16–25. IEEE Computer Society, 2005.

[28] Dario Bottazzi, Rebecca Montanari, and Alessandra Toninelli. Context-aware middleware for anytime, anywhere social networks. *Intelligent Systems, IEEE*, 22(5):23–32, Sept.-Oct. 2007.

[29] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Francois Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition), 2006. http://www.w3.org/TR/2006/REC-xml-20060816/.

[30] Dan Brickley and Richard V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, 2004. http://www.w3.org/TR/rdf-schema/.

[31] Dario Bruneo, Antonio Puliafito, and Marco Scarpa. *Mobile Middleware: Definition and Motivations*, chapter 7, pages 145–167. In [21], 2006.

[32] Licia Capra, Wolfgang Emmerich, and Cecilia Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, october 2003.

[33] Harry Chen. *An Intelligent Broker Architecture for Pervasive Context-Aware Systems*. PhD thesis, University of Maryland, Baltimore County, Baltimore MD, USA, 2004.

[34] Harry Chen, Filip Perich, Timothy W. Finin, and Anupam Joshi. Soupa: Standard ontology for ubiquitous and pervasive applications. In *MobiQuitous*, pages 258–267. IEEE Computer Society, 2004.

[35] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1.

[36] Elizabeth F. Churchill and Christine A. Halverson. Guest editors' introduction: Social networks and social networking. *IEEE Internet Computing*, 9(5):14–19, 2005.

[37] Michael J. Covington, Wende Long, Srividhya Srinivasan, Anind K. Dey, Mustaque Ahamad, and Gregory D. Abowd. Securing context-aware applications using environment roles. In *SACMAT*, pages 10–20, 2001.

[38] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In Morris Sloman, Jorge Lobo, and Emil Lupu, editors, *POLICY*, volume 1995 of *Lecture Notes in Computer Science*, pages 18–38. Springer, 2001.

[39] Anind K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. PhD thesis, Georgia Institute of Technology, Georgia, USA, 2000.

[40] Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[41] Anind K. Dey, Gregory Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human Computer Interaction (HCI) Journal*, 16:97–166, 2001.

[42] Xin Dong, Alon Y. Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Simlarity search for web services. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *VLDB*, pages 372–383. Morgan Kaufmann, 2004.

[43] Markus Endler and Douglas C. Schmidt, editors. *Proceedings of the ACM/IFIP/USENIX International Middleware Conference 2003*, volume 2672 of *Lecture Notes in Computer Science*. Springer, 2003.

[44] David Martin et al. OWL-S: Semantic Markup for Web Services. W3C Member Submission, November 2004. http://www.w3.org/Submission/OWL-S/.

[45] Dieter Fensel, Katia P. Sycara, and John Mylopoulos, editors. *The Semantic Web - ISWC 2003, Second International Semantic Web Conference, Sanibel Island, FL, USA, October 20-23, 2003, Proceedings*, volume 2870 of *Lecture Notes in Computer Science*. Springer, 2003.

[46] Marcus Foth. Facilitating social networking in inner-city neighborhoods. *IEEE Computer*, 39(9):44–50, 2006.

[47] Li Gong and Gary Ellison. *Inside Java(TM) 2 Platform Security: Architecture, API Design, and Implementation*. Pearson Education, 2003.

[48] Paul Grace and Gordon S. Blair. *Reflective Middleware*, chapter 14, pages 339–362. In [21], 2006.

[49] Paul Grace, Gordon S. Blair, and Sam Samuel. A reflective framework for discovery and interaction in heterogeneous mobile environments. *Mobile Computing and Communications Review*, 9(1):2–14, 2005.

[50] Philip D. Gray and Daniel Salber. Modelling and using sensed context information in the design of interactive applications. In Murray Reed Little and Laurence Nigay, editors, *EHCI*, volume 2254 of *Lecture Notes in Computer Science*, pages 317–336. Springer, 2001.

[51] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[52] Tao Gu, Hung Keng Pung, and Daqing Zhang. A service-oriented middleware for building context-aware services. *Journal of Network and Computer Applications*, 28(1):1–18, 2005.

[53] Terry Halpin. *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufmann, 2001.

[54] Karen Henricksen, Jadwiga Indulska, and Andry Rakotonirainy. Modeling context information in pervasive computing systems. In Friedemann Mattern and Mahmoud Naghshineh, editors, *Pervasive*, volume 2414 of *Lecture Notes in Computer Science*, pages 167–180. Springer, 2002.

[55] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, and Werner Retschitzegger. Context-awareness on mobile devices - the hydrogen approach. In *HICSS*, page 292, 2003.

[56] Jason I. Hong and James A. Landay. An infrastructure approach to context-aware computing. *Human-Computer Interaction (HCI) Journal*, 16(2-3), 2001.

[57] Ian Horrocks and James A. Hendler, editors. *The Semantic Web - ISWC 2002, First International Semantic Web Conference, Sardinia, Italy, June 9-12, 2002, Proceedings*, volume 2342 of *Lecture Notes in Computer Science*. Springer, 2002.

[58] Quentin Jones and Sukeshini A. Grandhi. P3 systems: Putting the place back into social networks. *IEEE Internet Computing*, 9(5):38–46, 2005.

[59] Lalana Kagal. *A Policy-Based Approach to Governing Autonomous Behavior in Distributed Systems*. PhD thesis, University of Maryland, Baltimore County, Baltimore MD, USA, 2004.

[60] Lalana Kagal, Timothy W. Finin, and Anupam Joshi. A policy language for a pervasive computing environment. In *POLICY* [10], pages 63–.

[61] John Keeney, Vinny Cahill, and Mads Haahr. *Techniques for Dynamic Adaptation of Mobile Services*, chapter 15, pages 363–384. In [21], 2006.

[62] Uwe Keller, Rubén Lara, Holger Lausen, Axel Polleres, and Dieter Fensel. Automatic location of services. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *ESWC*, volume 3532 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.

[63] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP*, pages 220–242, 1997.

[64] Michael Klein and Birgitta König-Ries. Combining query and preference - an approach to fully automatize dynamic service binding. In *ICWS*, pages 788–791. IEEE Computer Society, 2004.

[65] Matthias Klusch, Benedikt Fries, and Katia P. Sycara. Automated semantic web service discovery with owls-mx. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *AAMAS*, pages 915–922. ACM, 2006.

[66] Graham Klyne and Jeremy Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax, 10 February 2004. http://www.w3.org/TR/rdf-concepts/.

[67] Graham Klyne, Franklin Reynolds, Chris Woodrow, Hidetaka Ohto, Johan Hjelm, Mark H. Butler, and Luu Tran. Composite capability/preference profiles (cc/pp): Structure and vocabularies 1.0, 2004. http://www.w3.org/TR/CCPP-struct-vocab/.

[68] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. The web and social networks. *IEEE Computer*, 35(11):32–36, 2002.

[69] Ora Lassila and Deepali Khushraj. Contextualizing applications via semantic middleware. In *MobiQuitous* [11], pages 183–191.

[70] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW*, pages 331–339, 2003.

[71] Ramiro Liscano and Kaining Wang. A sip-based architecture model for contextual coalition access control for ubiquitous computing. In *MobiQuitous* [11], pages 384–392.

[72] Cecilia Mascolo, Licia Capra, and Wolfgang Emmerich. Middleware for mobile computing (a survey). In E. Gregori, G. Anastasi, and S. Basagni, editors, *Neworking 2002 Tutorial Papers*, volume 2497 of *Lecture Notes in Computer Science*, pages 20–58. Springer, 2002.

[73] Ryusuke Masuoka, Bijan Parsia, and Yannis Labrou. Task computing - the semantic web meets pervasive computing. In Fensel et al. [45], pages 866–881.

[74] Peter Mika. Flink: Semantic web technology for the extraction and analysis of social networks. *J. Web Sem.*, 3(2-3):211–223, 2005.

[75] Simon Miles, Juri Papay, Vijay Dialani, Michael Luck, Keith Decker, Terry R. Payne, and Luc Moreau. Personalised grid service discovery. *IEE Proceedings - Software*, 150(4):252–256, 2003.

[76] Alan L. Montgomery. Applying quantitative marketing techniques to the internet. *Interfaces*, 31(2):90–108, 2001.

[77] Hans W. Nissen, Manfred A. Jeusfeld, Matthias Jarke, Georg V. Zemanek, and Harald Huber. Managing multiple requirements perspectives with metamodels. *IEEE Software*, 13(2):37–48, 1996.

[78] Tommaso Di Noia, Eugenio Di Sciascio, and Francesco M. Donini. Extending semantic-based matchmaking via concept abduction and contraction. In Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nicholas Gibbins, editors, *EKAW*, volume 3257 of *Lecture Notes in Computer Science*, pages 307–320. Springer, 2004.

[79] Daniel Oberle. *Semantic Management of Middleware*, volume 1 of *Semantic Web And Beyond Computing for Human Experience*. Springer, 2006.

[80] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In Horrocks and Hendler [57], pages 333–347.

[81] Alex Pentland. Socially aware computation and communication. *IEEE Computer*, 38(3):33–40, 2005.

[82] Andrei Popovici, Andreas Frei, and Gustavo Alonso. A proactive middleware platform for mobile computing. In Endler and Schmidt [43], pages 455–473.

[83] Anand Ranganathan and Roy H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In Endler and Schmidt [43], pages 143–161.

[84] Oriana Riva. Contory: A middleware for the provisioning of context information on smart phones. In Maarten van Steen and Michi Henning, editors, *Middleware*, volume 4290 of *Lecture Notes in Computer Science*, pages 219–239. Springer, 2006.

[85] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubn Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web service modeling ontology. *Applied Ontology], publisher = IOS Press, number = 1, issue = 1 pages =77-106, year = 2005.*

[86] Manuel Román, Roy H. Campbell, and Fabio Kon. Reflective middleware: From your desk to your hand. *IEEE Distributed Systems Online*, 2(5), 2001.

[87] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[88] William N. Schilit. *A System Architecture for Context-Aware Mobile Computing*. PhD thesis, Columbia University, 1995.

[89] Albrecht Schmidt and Kristof van Laerhoven. How to build smart appliances. *IEEE Personal Communications*.

[90] Morris Sloman. Policy driven management for distributed systems. *Journal of Network Systems Management*, 2(4), 1994.

[91] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Proceedings of the Workshop on Advanced Context Modelling, Reasoning and Management as part of UbiComp 2004*, 2004.

[92] Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Cool: A context ontology language to enable contextual interoperability. In Jean-Bernard Stefani, Isabelle M. Demeure, and Daniel Hagimont, editors, *DAIS*, volume 2893 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2003.

[93] Loren G. Terveen and David W. McDonald. Social matching: A framework and research agenda. In *ACM Trans. Comput.-Hum. Interact.*, pages 401–434, 2005.

[94] Alessandra Toninelli, Jeffrey M. Bradshaw, Lalana Kagal, and Rebecca Montanari. Rule-based and ontology-based policies: Toward a hybrid approach to control agents in pervasive environments. In *Semantic Web and Policy Workshop, in conj. with ISWC 2005*, pages 47–54, 2005.

[95] Alessandra Toninelli, Antonio Corradi, and Rebecca Montanari. Semantic-based discovery to support mobile context-aware service access. *Computer Communications Journal, Special Issue on Mobility Management and Wireless Access*, 31(5):935–949, 2008.

[96] Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila. A semantic context-aware access control framework for secure collaborations in pervasive computing environments. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist,

Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 473–486. Springer, 2006.

[97] Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila. Proteus: A semantic context-aware adaptive policy model. In *POLICY*, pages 129–140. IEEE Computer Society, 2007.

[98] Gianluca Tonti, Jeffrey M. Bradshaw, Renia Jeffers, Rebecca Montanari, Niranjan Suri, and Andrzej Uszok. Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. In Fensel et al. [45], pages 419–437.

[99] Eddy Truyen. *Dynamic and Context-Sensitive Composition in Distribited Systems.* PhD thesis, Katholieke Universiteit Leuwen, Belgium, 2004.

[100] Andrzej Uszok, Jeffrey M. Bradshaw, Renia Jeffers, Niranjan Suri, Patrick J. Hayes, Maggie R. Breedy, Larry Bunch, Matt Johnson, Shriniwas Kulkarni, and James Lott. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *POLICY* [10], pages 93–.

[101] Andrzej Uszok, Jeffrey M. Bradshaw, Matt Johnson, Renia Jeffers, Austin Tate, Jeff Dalton, and J. Stuart Aitken. Kaos policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41, 2004.

[102] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller. Meteor-s wsdi: A scalable infrastructure of registries for semantic publication and discovery of web services. *Journal of Information Technology and Management, Special Issue on Universal Global Integration*, (6).

[103] Matthias Wagner, Thorsten Liebig, Olaf Noppens, Steffen Balzer, and Wolfgang Kellerer. Towards semantic-based service discovery on tiny mobile devices. In *Proceedings of the Workshop on Semantic Web Technology for Mobile and Ubiquitous Applications*, pages 90–101, Hiroshima, Japan, November 2004.

[104] Sui-Yu Wang, Yuanbo Guo, Abir Qasem, and Jeff Heflin. Rapid benchmarking for semantic web knowledge base systems. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science*, pages 758–772. Springer, 2005.

[105] Xiao Hang Wang, Da Qing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using owl. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops (PERCOMW 2004)*, page 18, Washington, DC, USA, 2004. IEEE Computer Society.

[106] Xiaohang Wang, Jin Song Dong, ChungYau Chin, SankaRavipriya Hettiarachchi, and Daqing Zhang. Semantic space: An infrastructure for smart spaces. *IEEE Pervasive Computing*, 3(3):32–39, 2004.

[107] Terry Winograd. Architectures for context. *Human Computer Interaction*, 16(2-4), 2001.

[108] Eiko Yoneki and Jean Bacon. *Openness and Interoperability in Mobile Middleware*, chapter 20, pages 487–518. In [21], 2006.

Appendix A

List of Publications

A.1 Journals and Magazines

Alessandra Toninelli, Antonio Corradi, and Rebecca Montanari.
Semantic-based discovery to support mobile context-aware service access.
Computer Communications Journal, Special Issue on Mobility Management and Wireless Access, 31(5): 935-949. Elsevier, 2008.

Dario Bottazzi, Rebecca Montanari, and Alessandra Toninelli.
A Semantic Context-Aware Middleware Level Solution to Support Anytime and Anywhere Social Networks.
IEEE Intelligent Systems, Special Issue on Social Computing, 22(5):23-31. IEEE Computer Society Press, 2007.

Antonio Corradi, Rebecca Montanari, and Alessandra Toninelli.
Adaptive Semantic Middleware for Mobile Environments.
Journal of Networks,2(1):36-47. Academy Publisher, 2007. Paolo Bellavista,

Antonio Corradi, Rebecca Montanari, and Alessandra Toninelli.
Context-Aware Semantic Middleware for Next Generation Mobile Systems.
IEEE Communications Magazine, Special Issue on Advances in Service Platform Technologies, 44(9): 62-71, IEEE Communications Society, 2006.

A.2 Chapters in International Books

Paolo Bellavista, Rebecca Montanari, Daniela Tibaldi, and Alessandra Toninelli.
Trust Management and Context-Driven Access Control.
Y.Zhang, J.Zheng, M.Ma Eds: Handbook of Research on Wireless Security., ISBN-13: 978-1599048994. Information Science Reference (January 15, 2008).

A.3 Conference Proceedings

Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila.
A Semantic Context-Aware Access Control Framework for Secure Collaborations in Pervasive Computing Environments.
Fifth International Semantic Web Conference (ISWC), LNCS 4273: 473-486. Springer, 2006.

Alessandra Toninelli, Rebecca Montanari, Lalana Kagal, and Ora Lassila.
Proteus: A Semantic Context-Aware Adaptive Policy Model.
IEEE 2007 International Workshop on Policies for Distributed Systems and Networks (POLICY), pp. 129-140. IEEE Computer Society Press, 2007.

Alessandra Toninelli, Rebecca Montanari, and Antonio Corradi.
Dynamic Configuration of Semantic-based Service Provisioning to Portable Devices.
Symposium on Applications and the Internet (SAINT), IEEE Computer Society Press, 2006.

Rebecca Montanari, Alessandra Toninelli, and Jeffrey M. Bradshaw.
Context-based Security Management for Multi-Agent Systems.
Symposium on Multi-Agent Security and Survivability (MAS&S), IEEE Conference Proceedings, 2005.

Antonio Corradi, Rebecca Montanari, Daniela Tibaldi, and Alessandra Toninelli.
A Context-Centric Security Middleware for Service Provisioning to Pervasive Environments.
Symposium on Applications and the Internet (SAINT), IEEE Computer Society Press, 2005.

A.4 Workshop Proceedings

Alessandra Toninelli, Lalana Kagal, Jeffrey M. Bradshaw, and Rebecca Montanari.
Rule-based and Ontology-based Policies: Toward a Hybrid Approach to Control Agents in Pervasive Environments.
Semantic Web and Policy Workshop, held with ISWC 2005, 2005.

Alessandra Toninelli, Rebecca Montanari, and Antonio Corradi.
Semantic Discovery for Context-Aware Service Provisioning in Mobile Environments.
First International Workshop on Managing Context Information in Mobile and Pervasive Environments, held with MDM 2005. CEUR-WS: 7 Dec 2005.

Antonio Corradi, Rebecca Montanari, and Alessandra Toninelli.
Adaptive Semantic Support Provisioning in Mobile Internet Environments.
International Workshop on Context-Aware Adaptation and Personalization for the Mobile Internet, held with SAINT 2005. IEEE Computer Society Press, 2007.