University of Zagreb
Faculty of Electrical Engineering and Computing
Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Alisa Devlić

Semantic agents for location-aware service provisioning in mobile network

Master thesis
Magistarski rad

Zagreb, 2005

The Master Thesis has been completed at the Department of Telecommunications of the Faculty of Electrical Engineering and Computing, University of Zagreb

Advisor: Gordan Ježić, Ph.D.
Assistant Professor, FER, University of Zagreb

Master thesis has 126 pages.

Thesis number:

The Master Thesis evaluation committee:

1. Ignac Lovrek, Ph.D., Professor, FER, University of Zagreb
2. Gordan Ježić, Ph.D., Assistant Professor, FER, University of Zagreb
3. Vlatko Čerić, Ph.D., Professor, Faculty of Economy, University of Zagreb

The Master Thesis defense committee:

1. Ignac Lovrek, Ph.D., Professor, FER, University of Zagreb
2. Gordan Ježić, Ph.D., Assistant Professor, FER, University of Zagreb
3. Vlatko Čerić, Ph.D., Professor, Faculty of Economy, University of Zagreb
4. Maja Matijašević, Ph.D., Professor, FER, University of Zagreb

Date of Master Thesis defense: December 20th, 2005

Acknowledgements

There is a number of people that have supported me during the process of writing this thesis. Firstly, I would like to thank my advisor Assistant Professor Gordan Ježić for his support, patience, and reducing my obligations during the last semester of thesis completion. I thank also Krunoslav Tržec for directing my research interest towards Semantic Web technologies, and supporting my efforts with insightful comments and arguments.

The majority of work presented in the thesis have been developed in the research project "Remote Operations Management" at the Department of Telecommunications in the cooperation with Ericsson Nikola Tesla company. I thank the members of the team for creating the research atmosphere and contributing with the comments to this work.

I am mostly grateful to my boyfriend, Alan Graf, for his consistent support and for all the nights he stayed awake with me to encourage me. And finally, I would like to thank my family, who always believed in me and supported me to pursue my goals.

Alisa Devlić

SEMANTIC AGENTS FOR LOCATION-AWARE SERVICE PROVISIONING IN MOBILE NETWORK

SEMANTIČKI AGENTI ZA PRUŽANJE USLUGA OVISNIH O LOKACIJI U POKRETNOJ MREŽI

The Master Thesis investigates the issues of service provisioning in mobile network. Service provisioning is defined as the setting in place and configuring of the hardware and software required for activating a telecommunications service for a customer. The thesis proposes and implements a solution for a flexible and efficient service provisioning using semantic agents in the mobile network. Semantic agents are intelligent software agents that collect user preferences for the required service, and discover the available advertised services offered by service providers. They use the implemented matchmaking algorithm in the thesis to determine from the available services the one that best meets user's requirements. When the service is determined and found on the Web, the user can install and invoke it on the mobile device.

The service that was utilized for service provisioning is the location-aware content delivery service that delivers personalized content to mobile users depending on their current location, utilized terminal and preferences. It consists of the client and server part, for which the provisioning is performed separately. The client part is provisioned on the described manner, using semantic agents. The server part of the service is migrated, installed and invoked on the remote node, using the multi-agent system supporting remote software maintenance operations in the network.

Alisa Devlić

SEMANTIČKI AGENTI ZA PRUŽANJE USLUGA OVISNIH O LOKACIJI U POKRETNOJ MREŽI

SEMANTIC AGENTS FOR LOCATION-AWARE SERVICE PROVISIONING IN MOBILE NETWORK

Magistarski rad istražuje probleme pružanja usluga u pokretnoj mreži. Pružanje usluge je definirano kao postavljanje i podešavanje sklopovlja i programske opreme potrebne za aktiviranje telekomunikacijske usluge korisniku. U radu je predložen i napravljen sustav za fleksibilno i efikasno pružanje usluga uporabom semantičkih agenata u pokretnoj mreži. Semantički agenti su inteligentni pokretni agenti koji prikupljaju korisničke preference za zahtijevanu uslugu, i otkrivaju raspoložive usluge objavljene od strane pružatelja usluga. Oni koriste algoritam za semantičko uspoređivanje, izrađen u radu, kako bi odredili koja od navedenih usluga najbolje odgovara korisnikovim zahtjevima. Kad je usluga određena i utvrđena njezina lokacija na Web-u, korisnik ju može instalirati i pokrenuti na svom pokretnom uređaju.

Usluga koja je korištena u svrhu demonstracije pružanja usluge u magistarskom radu je usluga za dostavu sadržaja ovisnog o lokaciji, čija je funkcionalnost dostava personaliziranog sadržaja pokretnim korisnicima ovisno o njihovoj trenutnoj lokaciji, korištenom terminalu i preferencama. Sastoji se od klijentskog i poslužiteljskog dijela, čije se usluge pružaju odvojeno i na različit način. Pružanje klijentskog dijela usluge je opisano na naveden način, koristeći semantičke agente. Poslužiteljski dio usluge se migrira, instalira i izvodi na udaljenom čvoru, koristeći višeagentski sustav koji podržava udaljene operacije održavanja programske opreme u mreži.

Contents

List of Figures **iii**

Introduction **1**

1 Semantic Web Vision **3**

1.1 Knowledge management 4

1.1.1 Ontologies 5

1.1.2 Logic 6

1.2 Semantic Web Architecture 7

1.3 Web Ontology Languages 8

1.3.1 RDF 10

1.3.2 RDF Schema 12

1.3.3 OWL 16

1.4 Semantic Web services 23

1.4.1 Overview of OWL-S 23

1.4.2 Service Process Model 25

2 Semantic agents **27**

2.1 Intelligent software agents 28

2.1.1 Agent definition 28

2.1.2 Agent classification 29

2.2 Semantic matchmaking 30

2.2.1 OWL-S Matchmaking algorithm 30

2.3 JADE agent platform 42

2.3.1 LEAP add-on 44

3 Context-aware services **47**

3.1 Definition of context 48

3.2 Context information 49

3.3 Location-aware services 50
3.3.1 Proposed location semantics for building higher-level context information 51

4 Location-aware content delivery system architecture 53
4.1 Requirements and Usage Scenarios 55
4.2 Reference architecture 59
4.2.1 Communication layer 61
4.2.2 Service layer 61
4.2.3 Application layer 62
4.2.4 Component interaction 63
4.3 System architecture 65
4.3.1 Service architecture for 3G mobile networks 68
4.3.2 Location-based publish/subscribe web service 70
4.3.3 User Profile Management Web service 73
4.3.4 User profile creation 75

5 System implementation 81
5.1 Implementation overview 81
5.2 Client application 82
5.3 User Profile Management Service Implementation 88
5.4 System deployment 92

6 Service provisioning 95
6.1 Service Provisioning in the Mobile Network 96
6.1.1 Over-The-Air Service Provisioning 97
6.2 Location-aware service provisioning 100
6.2.1 Client application provisioning using semantic agent 101
6.2.2 Server-side installation with RMS 108

7 Related work 113

Conclusion 119

Literature 121

Summary 127

Curriculum Vitae 129

List of Figures

1.1 A Semantic Web layer cake 7
1.2 RDF model 10
1.3 RDF generic model 11
1.4 Graph representation of a triple 12
1.5 RDQL query 16
1.6 OWL service ontology 25

2.1 OWL-S model of the location-aware content delivery Web service . 32
2.2 OWL-S nodes class diagram 34
2.3 Broker agent components 41
2.4 The software architecture of JADE Agent Platform 43
2.5 The communication architecture of JADE 44
2.6 The LEAP execution environment 45
2.7 Execution modes 46

4.1 Zagreb city area 56
4.2 Reference architecture 60
4.3 Current location-based subscription 64
4.4 Landmark-based subscription 65
4.5 Non location-based subscription 66
4.6 System architecture 67
4.7 3G service architecture 69
4.8 Location-based publish/subscribe web service 71
4.9 LBPS interface 72
4.10 User Profile Management 73
4.11 User Profile administration 74
4.12 ProfileMgmt interface 75
4.13 RDF graph 77

5.1 System implementation 82
5.2 MIDlet class diagram 83
5.3 Login and menu screen 84
5.4 Declare landmark 84
5.5 View visited landmarks 85
5.6 Modify availability 85
5.7 Get contacts 86
5.8 Create content 87
5.9 Subscribe screen 87
5.10 Content delivered via MMS and e-mail 88
5.11 Authentication 89
5.12 Main menu 89
5.13 Creating a new profile 90
5.14 User selection 91
5.15 View an existing profile 92
5.16 Deployment 93

6.1 A simplified view of OTA provisioning 97
6.2 A detailed view of OTA provisioning 98
6.3 OTA application provisioning lifecycle 100
6.4 Location-aware service provisioning 101
6.5 FIPA Contract Net interaction protocol 102
6.6 Service offerer agent ("alisa") started in the main container 103
6.7 Service requester agent started on mobile device 104
6.8 The service output on mobile device 107
6.9 The service installation and execution on mobile device 108
6.10 RMS concept 109
6.11 Remote maintenance operations with mobile agents 109
6.12 Graphical user interface 110
6.13 Remote Location Tasks dialog 111
6.14 Location-aware content delivery service in running state 112

Introduction

In the thesis an idea of service provisioning using semantic agents in mobile network has been investigated. It is applied to deployment of location-aware content delivery service, that delivers personalized content to mobile users depending on their current location, utilized terminal and preferences. The requirements and usage scenarios, that have guided the system design, are discussed, followed by its architecture proposal that could be easy deployed in the 3G mobile networks infrastructure, and a prototype implementation.

Semantic agents have been defined as a set of agents running around the Web and executing complex actions on behalf of their user(s). The concept of Semantic Web is based on an idea of dynamic, heterogeneous, shared knowledge sources providing machine-understandable content in a similar way to that in which information is shared on the World Wide Web. It will gradually evolve from the existing Web, adding the meaning to the information available. The existing World Wide Web will transform from a collection of static Web pages for browsing, to become a Web of interactive, automated, and intelligent services that interrelate via the Internet.

The problem with the existing Web services model is its inability to dynamically discover the most appropriate Web service that meets user demands. In the Semantic Web Vision, services are meant to be capable of being discovered, invoked, composed, and monitored automatically by software agents. A semantic agent, that is implemented in the thesis, solves the problem of semantically discovering required Web services, and compares the degree of similarity between the required and the advertised Web service(s) offered by service providers. If it finds that the advertised service matches the required service, it offers the user the ability to install this service and execute it on his mobile device.

The service provisioning of the location-aware content delivery service is performed separately for client and server part. Semantic agents are utilized for provisioning of the client application on the mobile device. They collect user preferences that the required service should meet, and discover the available advertised service(s) offered by service provider(s). Both services (the requested and the advertised service) are matched using the implemented semantic matchmaking algorithm, that assists an agent in finding the most appropriate service that meets user needs. When the service is determined and found on the Web, the user is offered a possibility to install and run the application on the mobile device.

The server part of the service is migrated, installed, and invoked on the remote node, using the multi-agent system supporting remote software maintenance operations in the network, without suspending or influencing its regular operation.

Thesis structure. The thesis is organized as follows: Chapter 1 describes the vision of Semantic Web, and gives a brief overview of its architecture and Web ontology languages. Special attention is given to the role of Semantic Web services in the Semantic Web vision, due to the need for them to be dynamically discovered, invoked, composed, and monitored, without human mediation. Semantic agents are intelligent software agents envisioned to run on the Web, performing complex actions on behalf of their users. They reason about the knowledge incorporated in Semantic Web services, and their concept is presented in Chapter 2. This chapter describes the concept of semantic matchmaking of Web services, followed by the precise explanation of the implemented algorithm. Chapter 3 deals with a theoretical background of context and context-aware services. It also gives a proposal of the user-understandable location semantics, that can gather and structure different formats of location information, to build higher-level context used by various location-aware services. Location-aware content delivery system architecture is presented in Chapter 4, that could be easy deployed in third generation mobile network infrastructure. Chapter 5 discusses the system implementation of the architecture proposed in the previous chapter. In Chapter 6 the idea of service provisioning in the mobile network is outlined. The recommended practice of Over-The-Air provisioning is described, and the new concept of provisioning applications on mobile devices using semantic agents is presented. The server part of the service is delivered on the target node in local area network using the Remote Maintenance Shell (RMS) system, a framework developed for remote software maintenance operations, explained in the same chapter. Chapter 7 gives an overview of the existing solutions and systems related to semantic matchmaking of Web services and location-aware computing. Their advances and shortcomings are analyzed and compared to solutions proposed in the thesis.

Chapter 1

Semantic Web Vision

Most of today's Web content is human-understandable. The information is presented in a satisfactory way, but what lacks is information about content. For machines to be more data-processable, the *meaning* of data has to be added to the content and its formatting information. The term *metadata* refers to such information: data about data. Therefrom derives the term *semantic* in Semantic Web.

The aim of Semantic Web is to represent Web content in a form that is more easily machine-processable and use intelligent techniques to take advantage of these representations. In the vision of Tim Berners-Lee, the initiator of the Semantic Web and inventor of the WWW in the late 1980s, the Semantic Web will gradually evolve from the existing Web, where the meaning of information will play an important role.

This chapter outlines a vision of Semantic Web and is organized as follows: Section 1.1 explains knowledge management issues and promises in the future of Semantic Web. In Section 1.2 a Semantic Web architecture overview is given. Section 1.3 briefly describes Web ontology languages, such as RDF, RDF Schema (RDFS), and OWL. Section 1.4 discusses the role of Semantic Web services in the Semantic web vision and their need to be dynamically discovered, invoked, composed, and monitored, without human mediation. A problem with the existing Web services model is to find the most appropriate service that meets user needs, meaning that user requests for a service need to be matched against the services advertised by service providers. An overview of OWL-S, an OWL-based language used to describe service capabilities, is given in Section 1.4.1. OWL-S allows describing both the user requested and service provider's advertised services.

1.1 Knowledge management

Knowledge management concerns itself with acquiring, accessing, and maintaining knowledge within an organization [AvH04]. Current knowledge management techniques suffer from the following limitations:

- *searching for information* depends on keyword-based search engines that retrieve a lot of irrelevant documents;
- *extracting information* involves human interaction to browse the retrieved documents for relevant information;
- *maintaining information* can cause problems, such as inconsistencies in terminology and failure to remove outdated information;
- *viewing information* with restriction to certain group of users cannot be realized over the Web.

The function of the Semantic Web relies on knowledge representation: computer-accessible structured collections of information and sets of inference rules that can be used to conduct automated reasoning.

The Semantic Web promises much more advanced knowledge management systems:

- knowledge will be organized in conceptual spaces according to its meaning;
- automated tools will support check for inconsistencies and knowledge extraction;
- semantic query answering will replace the traditional keyword-based search: knowledge will be retrieved, extracted and presented in a human-friendly form;
- query answering on several documents will be supported;
- defining who is authorized to view certain parts of information (even parts of documents) will be possible.

1.1.1 Ontologies

The term *ontology* (from the greek *ontologia*) originates from philosophy, namely the branch of metaphysics concerned with identifying the kinds of things that actually exist, and how to describe them. In the Information Science, an ontology is a hierarchical knowledge structure organized by subcategorizing things according to their essential (or at least relevant and cognitive) qualities. In the concept of Semantic Web, an ontology is defined as: *an explicit and formal specification of conceptualization*, meaning that it presents a formal description of concepts and relationships among them in some area of interest. Therefore, ontology is a terminology that provides a *shared understanding of a domain*, that can be communicated across people and application systems.

The most typical kind of ontology for the Web contains a taxonomy and a set of inference rules [BLHL01]. The taxonomy defines classes and relations among them. A large number of relations among entities can be expressed by assigning properties to classes and allowing subclasses to inherit such properties. The inference rules enable deriving new data from the data that is already known. A program could easily deduce, for instance, that a University of Zagreb, being in Zagreb, must be in Croatia, which is in the Europe, and being situated in Europe means being European. Therefore, University of Zagreb should be a part of European universities. The computer doesn't truly "understand" any of this information, but it can effectively manipulate terms in the human-understandable way.

Ontologies can enhance the functioning of the existing Web in many ways. They can be used to improve the accuracy of Web searches: the search program can only look for those pages that refer to a precise concept instead of all the ones containing ambiguous keywords. More advanced applications will use ontologies to relate an information on the page to the associated knowledge structures and inference rules. For example, the page could contain additional information about the desired subject, that would usually require a human to browse the various pages turned by a search engine. In a Semantic Web, this page will be linked to the ontology page that defines all the information about the particular subject, and is easily processed by a computer and queried by the user.

In addition, the ontologies' markup is more suitable to develop programs that can tackle complicated questions whose answers do not reside on the single Web page. Suppose an *intelligent agent's* (represented as software running on the Web and performing actions on behalf of a user) task is to find a person from a telecommunications conference, held last year in Zagreb, whose last name is "Cook", with first name not known, who worked for Ericsson company, and whose son is a student at a Stockholm University. The program would first search for all pages of people whose name is "Cook" (sidestepping all the pages relating to cooks, cooking, the Cook Islands and so forth), find the ones that mention working for Ericsson, and follow links to Web pages of their children to track down if any of them study at the specified university.

1.1.2 Logic

Adding logic to the Web is accomplished by using rules to make inferences, choosing courses of action and answering questions. A mixture of mathematical and engineering decisions complicates this task. The logic must be powerful enough to describe complex properties of objects but not so powerful that agents can be tricked by being asked to consider a paradox. For the Semantic Web to become expressive enough to be useful in a wide range of situations, it will become necessary to construct a powerful logical language for making inferences.

Logic (from ancient Greek *logos*, meaning reason) is the study of arguments. It is the discipline that studies the principle of reasoning - a set of premises that are examined and arranged so as to bring a conclusion. In general, logic offers, first, a set of *formal languages* for expressing knowledge. Secondly, it provides a *well-understood formal semantics* for describing the meaning of sentences without caring about how it can be deduced. Finally, automated reasoners can deduce (infer) conclusions from the given knowledge. They can uncover the implicit knowledge, as well as unexpected relationships and inconsistencies.

Semantic web vision predicts a complex semantic web comprised of a great number of small ontological components largely created of pointers to each other and developed by web users in the same way that web content is currently created [Hen01].

Semantic agents, which are the synonym for intelligent agents in the context of Semantic Web, will make use of all the technologies described:

- metadata will be used to identify and extract the information from Web sources;

- ontologies will assist in Web searches to interpret retrieved information, and to communicate with other agents;

- logic will be used for processing retrieved information and for drawing conclusions.

1.2 Semantic Web Architecture

Figure 1.1 shows the "layer cake" of the Semantic Web (according to Tim Berners-Lee), that describes main layers of the Semantic Web design and vision. The diagram depicts a Semantic Web architecture in which languages of increasing power are layered one of the top of the other. Each language both exploits the features and extends the capabilities of the layers below.

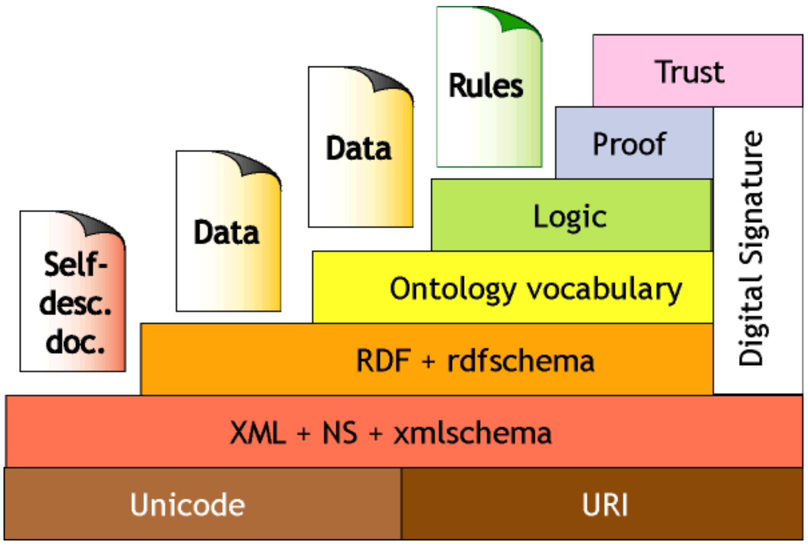


Figure 1.1: A Semantic Web layer cake

At the bottom of the architecture an XML is placed, a markup language for writing structured Web documents with a user-defined vocabulary. It allows anyone to design his own document format and then write a document in that format. It is particularly suitable for sending documents across the Web, because it enables data interchange between applications that both know about what the data is. XML provides a surface syntax for structured documents, but imposes no semantic constraints on the common interpretation of the data contained in these documents. This is a major limitation of XML: since XML just describes grammars, there is no way of recognizing the semantic unit from a particular domain of interest.

XML namespace is used to identify markup elements used in an XML document. The namespace points to an URI (Uniform Resource Identifier) where these elements and attributes are defined.

XML Schema is a language for restricting the syntax of XML applications. It represents a description of a type of an XML document, typically expressed in terms of constraints on the structure and content of documents of that type, thus extending the basic constraints imposed by XML itself.

The second layer is represented by RDF and RDF Schema (RDFS). Resource Description Framework (RDF) is a basic data model, like the entity-relationships model, for writing simple statements about Web objects (resources) and relations between them. The RDF data model does not rely on XML, but uses an XML-based syntax.

Just as XML Schema provides the vocabulary definition facility for XML, RDF Schema provides the similar facility for RDF. It provides modelling primitives for organizing Web objects into hierarchies. Key primitives are classes of RDF resources and properties, subclasses and subproperty relationships, and domain and range restrictions. It is based on RDF.

Web ontology languages are built on top of RDF(S), as the RDF Schema language is powerful enough to define more expressive languages on top of the limited primitives of RDF.

The logic layer is used to enhance the ontology language further and to allow the writing of application-specific declarative knowledge.

The proof layer involves the actual deductive process as well as representation of proofs in Web languages (from lower levels) and proof validation.

Digital signatures are another important feature of Semantic Web. They are encrypted blocks of data that computers and agents can use to verify that the attached information has been provided by a trusted source.

Finally, the trust layer will emerge through the use of digital signatures and other kinds of knowledge, based on recommendations by trusted agents or on rating and certification agencies and consumer bodies. Being located on the top of pyramid, trust is a very important concept in the Semantic Web: the overall success of the Semantic Web will depend on people's trust to data and quality of information provided.

1.3 Web Ontology Languages

Ontology languages allow users to write explicit, formal conceptualizations of domain models.

The main requirements for ontology languages are [AvH04]:

- a well-defined syntax
- a formal semantics
- efficient reasoning support
- sufficient expressive power
- convenience of expression

A *well-defined syntax* is important for machine-processing of information, becoming a necessary condition for all languages to fulfill.

A *formal semantics* unambiguously describes the meaning of knowledge. Clearly defined and well-understood semantics are essential if the ontology is to be used within a community for information exchange.

Semantics is a prerequisite for *reasoning support*. Reasoning support is important because it allows one to:

- check for inconsistencies of the ontology and the knowledge;
- check for unintended relationships between classes;
- automatically classify instances in classes.

Automated checking like the preceding ones are valuable for designing large ontologies, where multiple authors are involved, as well as for integrating and sharing ontologies from different sources.

In designing a powerful ontology language one should be aware of a trade-off between *sufficient expressive power* (what language can say) and *efficient reasoning support* (whether the language is computable in real-time). The richer the language is, the more inefficient the reasoning support becomes. Therefore a compromise should exist - a language that can be supported by efficient reasoners while being sufficiently expressive to design large ontologies.

The W3C has given a final approval to two key Semantic Web technologies, the revised Resource Description Framework (RDF) and Web Ontology Language (OWL), which will be briefly described in the following subsections.

1.3.1 RDF

The Resource Description Framework (RDF) [RDF03], developed by World-Wide Web Consortium (W3C), provides a framework for describing and exchanging metadata on the Web. It allows descriptions of Web resources to be made publicly available in a machine understandable form through the Uniform Resource Identifier (URI). With the defined semantics services can develop processing mechanisms to exchange information between applications.

The motivation for introducing RDF in the Semantic Web came from well-known limitations of XML markup language. Despite its characteristics for data interchange between applications, as well as providing parsers, it does not provide any semantics associated to the data.

The RDF basic concept is built upon describing resources through a collection of properties, called RDF Description, each of them consisting of property type and value (Fig. 1.2). Resources can be thought as "things" that will be talked about. They can be authors, books, publishers, places, people, hotels, rooms, etc. Every resource is assigned an URI. An URI can be an URL (Uniform Resource Locator, or Web address) or some other identifier that does not necessarily enable access to the resource.

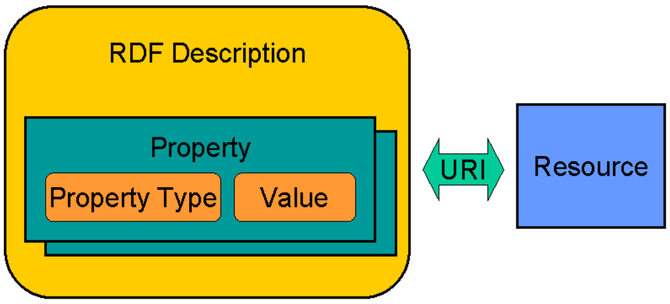


Figure 1.2: RDF model

RDF uses XML model and syntax to describe resources. It utilizes the XML Namespace facility, which points to an URI, to uniquely identify set of properties, commonly called the schema. Multiple namespaces can be used to provide properties in a single rdf document.

Properties are a special kind of resources, that describe relations between resources, for example "written by", "age", "title", and so on. Properties in RDF are also defined with URIs (in practice with URLs).

An RDF document consists of statements, combinations of a resource, a property and a value assigned to that property. Values can either be resources or literals (Fig. 1.3). Literals are atomic values (strings), the structure which is not further discussed.

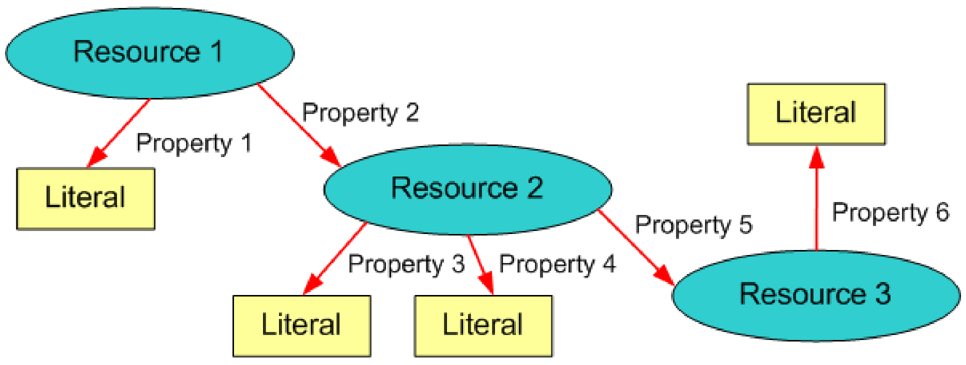


Figure 1.3: RDF generic model

RDF statements are written in triples: subject, predicate and object. The relationship of the RDF Schema with the RDF document is as follows: classes are mapped to subjects and predicates to properties. RDF statements present the main idea of creating semantics and describing relationships between objects, while the RDF Schema contains interpretations of information given in the statement.

An example of a statement is
Alisa Devlic is the author of the Master thesis.

The simplest way of interpreting this statement is to use the definition and consider the triple:
(http://www.tel.fer.hr/masterThesis.pdf, http://www.mydomain.org/author, "Alisa Devlic").

The triple (x, P, y) can be thought of as a logical formula P(x, y), where the binary predicate P relates the object x to the object y. RDF offers only binary predicates (properties). It is important to notice that the property "author" and one of the two objects are identified by URLs, whereas the other object is simply identified by string.

The graph representation of the correspondent statement is presented on Fig. 1.4. An RDF graph consists of nodes represented as ovals and contain their RDF URI references where they have them, all the predicate arcs labelled with RDF

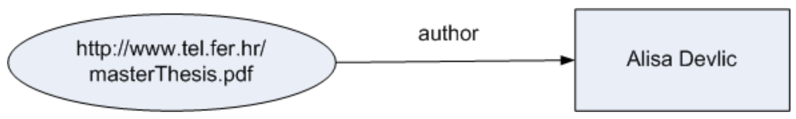


Figure 1.4: Graph representation of a triple

URI references and plain literal nodes written in rectangles [rdf04c]. Graphs are a powerful tool for human understanding. But the Semantic Web vision requires machine-accessible and machine-processable representations.

Therefore, the third and the official RDF specification defines an XML representation of RDF and the following serialization represents the given statement:

```
<?xml version="1.0" encoding="UTF-16"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:mydomain="http://www.mydomain.org/my-rdf-ns">

<rdf:Description
    rdf:about="http://www.tel.fer.hr/masterThesis.pdf">
    <mydomain:author>Alisa Devlic</mydomain:author>
</rdf:Description>

</rdf:RDF>
```

The RDF document begins with an XML element with the tag *rdf:RDF* [AvH04]. The content of this element is a number of descriptions, that use *rdf:Description* tags. Every description makes a statement about a resource, identified in one of the three possible ways:

- using an *about* attribute to reference an existing attribute;
- an *ID* attribute to create a new resource;
- without a name, creating an *anonymous* resource.

The *rdf:Description* element in the example above makes a statement about the resource *http://www.tel.fer.hr/masterThesis.pdf.* Within the resource, the property *author* is used as a tag, and the value of the property is represented by a literal *"Alisa Devlic"*.

1.3.2 RDF Schema

RDF doesn't have a built-in mechanism for defining properties and describing relationships between resources and properties. That is the task of the RDF Schema

(RDFS). It specifies how to use RDF to describe RDF vocabularies. This specification provides modelling primitives for expressing information in the Web.

Resources are defined as instances of one or more classes that can contain properties and they are described with the hierarchy of classes and subclasses.

Classes are themselves resources that are often identified with RDF URI references and may be described using RDF properties.

The core classes are [rdf04a]:

- *rdfs:Resource*, the class of all resources;
- *rdfs:Class*, the class of all classes;
- *rdfs:Literal*, the class of literal values such as strings and integers;
- *rdfs:DataType*, the class of RDF datatypes; each instance of *rdfs:DataType* is a subclass of *rdfs:Literal*;
- *rdf:XMLLiteral*, the class of XML literals values, and an instance of *rdfs:DataType*;
- *rdf:Property*, the class of all properties.

Properties are defined with domain and range of classes. The domain of a property refers to the class the property belongs to, and it is not restricted to a single class. The range represents the type of values the property can assume, and that can be simple data types or further classes having its own properties.

Core properties for defining relationships are:

- *rdf:type*, that relates resource to its class;
- *rdfs:subClassOf*, that relates a class to one of its superclasses; all instances of a class are subclasses of its superclass;
- *rdfs:subPropertyOf*, that relates a property to one of its superproperties.

An example is provided stating that all assistants are faculty members:

```
<rdfs:Class rdf:about="assistant">
    <rdfs:subClassOf rdf:resource="facultyMember"/>
</rdfs:Class>
```

Often it is useful to provide more information about a resource for human readers with the following properties:

- *rdfs:label*, associates a human-friendly name (label) to a resource;
- *rdfs:comment*, associates comments, typically longer text to a resource.

Core properties for restricting properties are:

- *rdfs:domain*, that specifies a domain of a property;
- *rdfs:range*, that specifies a range of a property.

Here is an example, stating that lab exercises apply to assistants only and that their value is always a literal.

```
<rdf:Property rdf:ID="labExercise">
    <rdfs:comment>
     It is a property of assistants and takes literals as values.
    </rdfs:comment>
    <rdfs:domain rdf:resource="#assistant"/>
    <rdfs:range rdf:resource="&rdf;Literal"/>
</rdf:Property>
```

Utility properties used to describe resources and define links to resources on the Web are:

- *rdfs:seeAlso*, relates resource to another resource that explains it;
- *rdfs:isDefinedBy*, relates a resource to the place where its definition, typically RDF Schema, is defined;
- *rdf:value*, idiomatic property used to describe structured values.

Container classes and properties are used to represent collections. Three different kinds of containers are defined, as well as the RDF container class itself:

- *rdf:Bag*, the class of bags used to indicate that properties in the container are intended to be unordered;
- *rdf:Seq*, the class of sequences used to indicate the numerical ordering of the container membership properties;
- *rdf:Alt*, the class of alternatives used to indicate that one of the members of the container will be selected;
- *rdfs:Container*, the superclass of all container classes including the three preceding ones.

The class *rdf:ContainerMembershipProperty* has as instances container membership properties, that are used to state that a resource is member of the container.

Container membership properties are referenced with the property *rdfs:member*, that represents a superproperty of all container membership properties.

Reification classes and properties define a vocabulary for describing RDF statement without stating them.

- *rdf:Statement*, the class of all reified statements;
- *rdf:subject*, that relates a reified statement to its subject;
- *rdf:predicate*, that relates a reified statement to its predicate;
- *rdf:object*, that relates a reified statement to its object.

Querying in RDQL

RDQL (RDF Data Query Language) has been implemented in a number of RDF systems for extracting information from RDF graphs. Therefore it is referenced as a query language for RDF. The reason why a new query language is introduced instead of using XML query language lies in the fact that XML is located at a lower level of abstraction than RDF.

XPath is a language for addressing and querying XML documents [xpa99]. Since XML can have different representations of a statement, different XPath queries would have to be written for each form [AvH04]. A better solution is to write queries at the level of RDF.

Currently, there is no standardized query languages for RDF and RDFS. RDQL has a status of a member submission at W3C [rdq04], meaning that it is still a subject to change.

An RDF model is a graph, often represented in triples. RDQL preserves a graph pattern, expressed as a list of triple patterns. Each triple pattern consists of named variables and RDF values (URIs and literals). An RDQL query can additionally have a set of constraints on the values of those variables, and a list of the variables required in the answer set. An example of RDQL query is provided on the Fig. 1.5.

This query matches all statements in the graph that have the predicate *http://www.w3.org/1999/02/22-rdf-syntax-ns#type* and the object *http://example.com/someType*. The variable "?x" will be bound to the label of the subject resource, and all such "x" will be returned as a result.

```
SELECT ?X
    WHERE (?X, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,
    <http://example.com/someType>)
```

Figure 1.5: RDQL query

RDQL was first released in Jena 1.2.0. Since then the following systems are known that provide RDQL:

- Jena [Jen]
- RDFStore [rdf04b]
- Sesame [ses05]
- PHP XML Classes [php]
- 3Store [3st04]
- RAP - RDF API for PHP [rap04]

In addition, RDQL is one language used for remote query by the Joseki RDF Server [jos03]. Joseki is a server for publishing RDF models on the web.

1.3.3 OWL

The expressivity of RDF and RDF Schema, defined as a measure of the range of constructs that can be use to formally, flexibly, explicitly, and accurately describe the components of ontology, is very limited. RDF is limited to binary predicates and RDF Schema is limited to a subclass hierarchy and a property hierarchy, with domain and range definitions of these properties. Therefore a need for a more powerful ontology modelling language has appeared.

This led to a joint initiative of a number of research groups from the United States and Europe to define a richer language, called DAML+OIL [dam01]. The name is a join of the U.S. proposal DAML-ONT [dam00] and the European language OIL [oil00].

DAML+OIL was a starting point for the W3C Web Ontology Working Group in defining OWL, a language that would be broadly accepted and standardized ontology language in Semantic Web. OWL is developed as a vocabulary extension of RDF and is derived from the DAML+OIL Web Ontology Language.

OWL is designed for use by applications that need to process the content of information instead of just presenting information to humans [owl04]. It facilitates greater machine interpretability of Web content than that is supported by XML, RDF, RDF Schema by offering additional vocabulary along with a formal semantics.

Ideally, OWL would be an extension of RDF Schema, where RDF meaning of classes and properties would be preserved, and additional language primitives to support the richer expressiveness required would be added. Such an extension would be consistent to the layered structure of Semantic Web architecture, but would work against obtaining expressive power and efficient reasoning.

To fulfill the full set of requirements for an ontology language, OWL is defined as three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full, each geared towards fulfilling different aspects of this requirements set.

OWL Lite

OWL Lite [owl02] presents a subset of the full OWL language constructors, having a few limitations. Unlike the full OWL language (and DAML+OIL), classes can only be defined in terms of named superclasses and only certain kinds of restrictions can be used. Equivalence on classes, and subclass between classes are allowed only on named classes. Similarly, property restrictions in OWL Lite use named classes. OWL Lite also has a limited notion of cardinality - it only permits cardinality values of 0 and 1.

The advantage of this sublanguage is that it is easier to grasp (for users) and easier to implement (for tool builders). The disadvantage is its restricted expressivity.

OWL DL

OWL DL (short for Description Logic) [owl04] is a sublanguage of OWL Full that offers the users maximum expressiveness, while retaining computational completeness (all conclusions are guaranteed to be computed) and decidability (all computations will finish in time) of reasoning systems. OWL DL includes all OWL language constructs, but they can be used only under certain restrictions (for example, while a class may be a subclass of many classes, a class cannot be an instance of another class).

OWL DL is so named due to its correspondence to description logics [BCM+02], a field of research that has studied a particular decidable fragment of first order logic. OWL was designed to support the existing Description Logic and has desirable computational properties for reasoning systems. The disadvantage is that a full compatibility with RDF is lost: an RDF document needs to be extended in some ways and restricted in others to become a legal OWL document. But, every legal OWL DL document is a legal RDF document.

OWL Full

The entire language is called OWL Full and uses all the OWL language primitives. It also allows combining these primitives in arbitrary ways with RDF and RDF Schema, enabling the possibility to augment the meaning of the predefined (RDF or OWL) vocabulary by applying the language primitives to each other. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual in its own right.

The advantage of OWL Full is its full upward compatibility with RDF, both syntactically and semantically. But it is unlikely that any reasoner will support every feature of OWL Full, thus imposing no computational guarantees. The language has become so powerful to be undecidable, without any hope of complete (or efficient) reasoning support.

Ontology developers should consider to adapt the OWL sublanguage that best suits their needs. Each of these languages is an extension of its simple predecessor, both in what can be legally expressed and in what can be validly concluded. The following set of relations hold:

- Every legal OWL Lite ontology is a legal OWL DL ontology;
- Every legal OWL DL ontology is a legal OWL Full ontology;
- Every valid OWL Lite conclusion is a valid OWL DL conclusion;
- Every valid OWL DL conclusion is a valid OWL Full conclusion.

The choice between OWL Lite and OWL DL depends on the extent to which users require the more expressive constructs provided by OWL DL. The choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modelling facilities of RDF Schema (e.g. defining classes of classes, or attaching properties to classes). When using OWL Full as compared to OWL DL, reasoning support is less predictable since complete OWL Full implementations will not be possible.

OWL syntax

OWL builds on RDF and RDF Schema and uses RDF's XML syntax. The OWL syntax will be explained on simple examples.

OWL documents are commonly called OWL ontologies and are RDF documents. An OWL document consists of optional ontology headers (generally at most one), plus a number of classes and properties definitions and facts about individuals.

The header of OWL ontology begins with an *rdf:RDF* element, that specify the number of namespaces.

```
<rdf:RDF
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-synatx-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
```

Classes are defined using an *owl:Class* element. The following representation defines a class Professor as follows:

```
<owl:Class rdf:ID="Professor">
    <rdfs:subClassOf rdf:resource="#AcademicStaff"/>
</owl:Class>
```

This way the class Professor is defined as a subclass of the class AcademicStaff. To emphasize that this class is disjoint from the assistantProfessor class, an *owl:disjointWith* element can be used:

```
<owl:Class rdf:about="#Professor">
    <owl:disjointWith rdf:resource="#AssistantProfessor"/>
</owl:Class>
```

Equivalence of classes can be expressed with *owl:equivalentClass* element:

```
<owl:Class rdf:ID="FacultyMember">
    <owl:equivalentClass rdf:resource="#AcademicStaff"/>
</owl:Class>
```

Two OWL class identifiers are predefined, namely the classes *owl:Thing* and *owl:Nothing*. The former is the most general class, representing a set of all individuals, and the latter is the empty set.

An enumeration class description is used in OWL to list individuals that are the instances of the class. It is defined with *owl:oneOf* property. The list of individuals is usually represented with RDF construct *rdf:parseType="Collection"*, for a set of list elements:

```
<owl:Class rdf:ID="Continents">
    <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Europe"/>
        <owl:Thing rdf:about="#Asia"/>
        <owl:Thing rdf:about="#Africa"/>
        <owl:Thing rdf:about="#NorthAmerica"/>
        <owl:Thing rdf:about="#SouthAmerica"/>
        <owl:Thing rdf:about="#Australia"/>
        <owl:Thing rdf:about="#Antarctica"/>
    </owl:oneOf>
</owl:Class>
```

OWL distinguishes between two types of properties when building an ontology:

- *Object properties*, that link individuals to other individuals;
- *Datatype properties*, that link individuals to datatype values.

An object property is defined as an instance of the built-in OWL class *owl:ObjectProperty*. Here is an example of an object property:

```
<owl:ObjectProperty rdf:ID="isLedBy">
    <rdfs:domain rdf:resource="#labExercise"/>
    <rdfs:range rdf:resource="#assistant"/>
</owl:ObjectProperty>
```

A datatype property is defined as an instance of the built-in OWL class *owl:DatatypeProperty*. Here is an example of an datatype property:

```
<owl:DatatypeProperty rdf:ID="name">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
```

Both *owl:ObjectProperty* and *owl:DatatypeProperty* are subclasses of the RDF class *rdf:Property*. RDF Schema constructs, such as: *rdfs:subPropertyOf*, *rdfs:subClassOf*, *rdfs:domain* and *rdfs:range*, can be applied to an OWL document.

Equivalence of properties can be defined through the use of the element *owl:equivalentProperty*:

```
<owl:ObjectProperty rdf:ID="lectures">
    <owl:equivalentProperty rdf:resource="#teaches"/>
</owl:ObjectProperty>
```

OWL also provides a construct *owl:inverseOf* to define an inverse relation between properties:

```
<owl:ObjectProperty rdf:ID="leads">
    <owl:inverseOf rdf:resource="#isLedBy"/>
</owl:ObjectProperty>
```

Property restrictions are a special kind of class description, describing a class of all individuals that satisfy a certain restriction. OWL distinguishes two types of property restrictions: value constraints and cardinality constraints.

- A value constraint puts constraints on the range of the property when applied to this particular class description. Value constraints are: *owl:allValuesFrom*, *owl:someValuesFrom*, and *owl:hasValue*;
- A cardinality constraint puts constraints on the number of values a property can take. OWL provides three constructs for restricting the cardinality of properties locally within a class context: *owl:maxCardinality*, *owl:minCardinality*, and *owl:cardinality*.

owl:allValuesFrom is used to specify the class of possible values the property specified by *owl:onProperty* can take. A simple example declares that all undergraduate courses are led by academic staff members:

```
<owl:Class rdf:about="#UndergraduateCourse">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isLedBy"/>
            <owl:allValuesFrom rdf:resource="#AcademicStaff"/>
        </owl:Restriction>
    </rdfs>subClassOf>
</owl:Class>
```

owl:someValuesFrom is used to specify the class of individuals for which at least one value of the property concerned is an instance of the class description or a data value in the data range. The following example declares that all academic staff members must teach at least one undergraduate course:

```
<owl:Class rdf:about="#AcademicStaff">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#teaches"/>
            <owl:someValuesFrom rdf:resource="#UndergraduateCourse"/>
        </owl:Restriction>
    </rdfs>subClassOf>
</owl:Class>
```

owl:hasValue states a specific value that the property specified by *owl:onProperty* must have. The following example declares that NetworkMobilityCourse is taught in the nineth semester.

```
<owl:Class rdf:about="#NetworkMobilityCourse">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#isTaughtIn"/>
            <owl:hasValue rdf:resource="#9thSemester"/>
        </owl:Restriction>
    </rdfs>subClassOf>
</owl:Class>
```

owl:maxCardinality describes a class of all individuals that have *at most N* semantically distinct values for the property concerned, where N is the value of the cardinality constraint. Similarly, *owl:minCardinality* describes a class of all individuals that have *at least N* semantically distinct values for the property concerned. For example, it can be declared that a department needs to have at least ten and at most fifty members:

```
<owl:Class rdf:about="#Department">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasMember"/>
            <owl:minCardinality
            rdf:datatype="&xsd;nonNegativeInteger">10</owl:minCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasMember"/>
            <owl:maxCardinality
            rdf:datatype="&xsd;nonNegativeInteger">50</owl:maxCardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

A restriction containing *owl:cardinality* constraint describes a class of all individuals that have *exactly N* semantically distinct values for the property concerned, where N is the value of the cardinality constraint. The following example declares that a PhD student must have exactly two supervisors:

```
<owl:Class rdf:about="#PhDStudent">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasSupervisor"/>
            <owl:cardinality
            rdf:datatype="&xsd;nonNegativeInteger">2</owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

Some properties of property elements can be defined directly:

- *owl:TransitiveProperty* defines a transitive property, such as "has better grade than", "is taller than", or "is parent of";
- *owl:SymmetricProperty* defines a symmetric property, such as "has same grade as";
- *owl:FunctionalProperty* defines a property that can take at most one value, such as "age" or "height";
- *owl:InverseFunctionalProperty* defines a property for which two different individuals cannot have the same value, for example "isTheSocialSecurityNumberFor".

It is possible to use Boolean combinations (union, intersection and complement) of classes in OWL ontology, expressed with the following constructs: *owl:unionOf*, *owl:intersectionOf* and *complementOf*. Boolean combinations can be nested arbitrarily. The following example defines administrativeStaff to be those staff members that are neither faculty nor technical support staff:

```
<owl:Class rdf:ID="AdminStaff">
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#StaffMember"/>
        <owl:Class>
            <owl:complementOf>
                <owl:Class>
                    <owl:unionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Faculty"/>
                        <owl:Class rdf:about="#TechSupportStaff"/>
                    </owl:unionOf>
                </owl:Class>
            </owl:complementOf>
        </owl:Class>
    </owl:intersectionOf>
</owl:Class>
```

1.4 Semantic Web services

World Wide Web is transforming from a collection of static web pages for browsing, to become a web of interactive, automated, and intelligent services that interoperate through the Internet. Multiple web services will interoperate to perform tasks, provide information, transact business, and generally take action for users, dynamically and on demand. That is very important for conducting business faster and more efficiently than before.

Today web services are discovered and invoked by human users, limiting the ability to dynamically discover and interact with each other autonomously, thus reducing the need for human mediation.

The problem that arises with the existing web services model is to match user requestors with service providers, in order to find the most appropriate service that meets users' needs. The problem is even bigger because the amount of information and available services on the web grows rapidly every day.

1.4.1 Overview of OWL-S

OWL-S provides a set of constructs for modelling Web services to be machine-interpretable. These constructs are defined in OWL language.

OWL-S is expected to accomplish the following tasks:

- *Automated Web service discovery* - is an automated process for location of Web services that can provide a particular class of service capabilities, that adhere to client-specified constraints. Rather than having user to manually conduct search for a service, the information describing the service could be expressed in OWL-S and a search agent could then read and interpret such descriptions ("advertisements") and find a suitable match;

- *Automated Web service invocation* - is the automatic invocation of a Web service by a computer program or an agent, with given only an ontology description of the service. To do the specified, an agent has to understand the inputs required from the service, the outputs provided, and how to execute the service automatically. OWL-S provides a declarative API for Web services that includes semantic of the arguments to be specified when executing the calls to services, as well as the semantics of what is returned in messages when services succeed or fail;

- *Automated Web service composition and interoperation* - involves the automated selection, composition, and interoperation of Web services to perform some more complex task. This process needs to be automated, which implies that the prerequisites and consequences of individual services have to be available to the composing agent.

OWL-S is a OWL-based Web service ontology, which has been developed to provide a core set of markup language constructs for describing the properties and capabilities of Web services. The OWL-S (formerly DAML-S) provides three types of knowledge associated to a Web service: Service Profile (what the service does), Service Model (how the service works), and Service Grounding (how to use the service). The structure of OWL ontology is presented on Fig. 1.6, consisting of four classes each aimed to enable execution of one of the main Web service tasks.

The class *Service* provides an entry point for any Web service description. One instance of *Service* will exist for each published Web service. Properties *presents*, *supports*, and *describedby* are properties of *Service*. Classes *ServiceProfile*, *ServiceModel*, and *ServiceGrounding* present ranges of these properties, respectively. Each of these classes provides an essential type of knowledge about particular service, as it is explained below.

The *Service Profile* provides the information needed for an agent to discover the service and to decide whether the service fulfills its demands. More specifically,

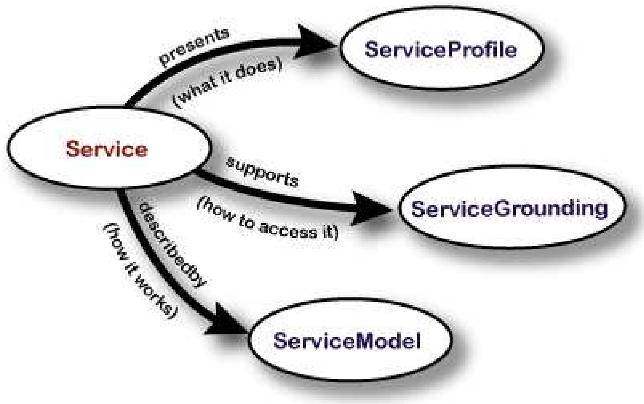


Figure 1.6: OWL service ontology

it specifies capabilities of the Web service by stating the inputs that are expected by the Web service as well as the outputs produced by it.

The *Service Model*, on the other hand, gives a description how the service works by decomposing the service into consistent sub-processes. It describes how to request the service and what outcomes will occur when the service is carried out. The exact functionality is presented through a process model (i.e. through the Process ontology and Process Control ontology).

Finally, the *Service Grounding* specifies the details how an agent can access the service. Typically, the grounding specifies the communication protocol, message formats, and other service-specific details, such as used port number in contacting the service.

The upper ontology for services specifies only two cardinality constraints: a service can be described by at most one service model, and a grounding must be associated with exactly one service.

Due to the understanding of implementation of the matchmaking algorithm, that is explained in the next section, the *Service Model* will be further elaborated.

1.4.2 Service Process Model

OWL-S distinguishes three types of processes: *atomic*, *simple*, and *composite*. *Atomic* processes correspond to operations the provider can perform directly, in a single interaction. They have no subprocesses and execute in a single step. *Simple* processes are used as elements of abstraction. They can be thought of as having single-step executions, but are not invocable. *Composite* processes are collections of processes (either atomic or simple) organized on the basis of some control flow structure that is specified by control constructs.

The OWL-S specifies the following control constructs:

- *Sequence*
- *Split*
- *Split+Join*
- *Choice*
- *Any-Order*
- *If-Then-Else*
- *Iterate*
- *Repeat-While*
- *Repeat-Until*

The *Sequence* construct defines a composite process whose subprocesses are executed one after the other; the components of the *Split* process are a bag of processes to be executed concurrently; with *Split+Join* some of component subprocesses are joined to be executed concurrently, thus having a partial synchronization; the *Any-Order* construct allows the process components to be executed in some unspecified order, but not concurrently; the *Choice* construct is used for non-deterministic choices between alternative flows; the *If-Then-Else* class is used for conditional expressions; the *Iterate* construct does not define how iterations are made or when they are initiated, terminated, or resumed. It serves as an abstract class for *Repeat-While* and *Repeat-Until* constructs. Both of these constructs iterate until a condition becomes false or true, following familiar programming language conventions.

Chapter 2

Semantic agents

Agents are pieces of software that work autonomously and proactively. They perform tasks on behalf of a user or other agent. Intelligent agents are agents that incorporate some reasoning or planning. Agents can be viewed as a model for distributed intelligence or a new model for developing software to interact over a network. Conceptually they evolved out of the concepts of object-oriented programming and component-based software development.

The Sematic Web is based on the idea of dynamic, heterogeneous, shared knowledge sources providing machine-understandable content in a similar way to that in which information is shared on the World Wide Web. To this extent, agents could use this knowledge to achieve their own goals, producing new knowledge that could be disseminated or published within a common framework. The effectiveness of such software agents will increase exponentially when more machine-readable Web content and automated services (including agents) will be available.

Semantic agents represent a term envisioned for a set of agents running around the Web, being able to perform complex actions on behalf of their users. In completion of its actions, an agent will receive some tasks and preferences from the user, seek information from Web sources, communicate with other agents, compare information about user requirements and preferences, select certain choices, and give answers to the user [AvH04].

This chapter is organized as follows: Section 2.1 deals with intelligent software agents, also called semantic agents, that reason about the knowledge incorporated in Semantic Web services. Section 2.1.1 gives a definition of an agent, while an agent classification is outlined in Section 2.1.2. Section 2.2. describes a concept of semantic matchmaking, that is realized through the implementation of an OWL-S matchmaking algorithm, fully explained in Section 2.2.1. Section 2.3 introduces

JADE, an agent platform chosen for implementation of semantic agents, and its add-on for mobile devices, LEAP, discussed in Section 2.3.1.

2.1 Intelligent software agents

2.1.1 Agent definition

The general definition of the agent, accepted in the thesis is [Her96]:
An agent is defined as a hardware or (more usually) software-based computer system that enjoys the following properties:

- *autonomy* - agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state;
- *social ability* - agents interact with other agents (and possibly humans) via some kind of agent communication language;
- *reactivity* - agents perceive their environment (which may be the physical world, a user via graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
- *pro-activeness* - agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative;
- *temporal continuity* - agents are continuously running processes (either running active in the foreground or sleeping/passive in the background), not once-only computations or scripts that map a single input and then terminate;
- *goal orientedness* - an agent is capable of handling complex, high-level tasks. The decision how such a task is best split-up in smaller subtasks, in which order and in which way these subtasks should be best performed, should be made by an agent itself.

This group of characteristics give a global impression of what an agent is and are connected to the weak notion of the concept "agent".

Intelligent software agents are a popular research object nowadays in the fields of psychology, sociology, and computer science. Agents are most intensely studied in the discipline of Artificial Intelligence (AI). Therefore the AI research community has given a "stronger" and more specific definition of an agent than that is sketched out above. In addition of having the previously defined properties, an agent is meant to have more human-like characteristics, such as: knowledge, belief, intention, emotion, and obligation.

Agents that fit into the stronger notion of agent have one or more following characteristics:

- *mobility* - the ability of an agent to move around an electronic network;
- *benevolence* - the assumption that agents do not have conflicting goals, and that every agent will therefore always try to do what is asked of it;
- *rationality* - the assumption that an agent will act in order to achieve its goals and will not act in such a way as to prevent its goals to be achieved, at least insofar as its beliefs permit;
- *adaptivity* - an agent should be able to adjust itself to the habits, working methods and preferences of its user;
- *collaboration* - an agent should not unthinkingly accept (and execute) instructions, but should take into account that the human user makes mistakes (e.g. give an order that contains conflicting goals), omits important information, and/or provides ambiguous information. For instance, an agent should check things by asking questions to user, or use a built-up user model to solve problems like these. An agent should even be allowed to refuse to execute certain tasks, because (for instance) they would put an unacceptably high load on the network resources or because it would cause damage to other users.

2.1.2 Agent classification

Agents may be usefully classified according to the subset of the properties that they enjoy. They can be further classified according to the tasks they perform, for example as [Lov99]:

- information;
- cooperation;
- transaction agents.

Information agents provide operations of information retrieval and filtering, as well as information providing. Cooperation agents usually cooperate together to solve some complex problem. Transaction agents perform specific tasks in business process management, such as transaction handling and supervision in e-commerce, manufacturing, etc.

2.2 Semantic matchmaking

In the Semantic Web vision, services are meant to be capable of being discovered, invoked, composed, and monitored automatically by software agents. In the thesis a semantic matchmaking agent will be presented, i.e. an intelligent software agent that first solves the problem of semantically discovering required Web services and then compares the discovered service with advertised services, in order to find the most appropriate service that meets user's needs [MDT05]. It requires the user's requested service and service provider's advertised services to be described in OWL-S, an OWL-based language for describing service capabilities, and it uses the implemented matchmaking algorithm to perform a semantic matchmaking of user requests with advertisements from service providers.

The terminology used in the thesis consists of entities that make use of Web services: *service requester* and *service provider*.

The entity that seeks to invoke an external Web service in order to fulfill its task is called *service requestor*. The service that the requestor seeks to invoke is referred to as the *requested service*. The requested service is not an actual service, but an ideal abstraction of a Web service.

Any entity that provides a Web service is referred to as the *service provider* and the Web service he is offering is called the *advertised service*. Ideally, if it happens that the advertised service is exactly what the requester was looking for, then the advertised service exactly matches the required service.

2.2.1 OWL-S Matchmaking algorithm

The goal of the matchmaking algorithm is to assist a software agent in selecting the most suitable Web service for given user preferences.

The algorithm will decide if a Web service, based on the available semantic information encoded in OWL-S, fulfills requirements specified by the user in the requested service. More specifically, it will match advertised service parameters with requested capabilities (parameters). This match will result with some matching degree, a ranking result. Such a ranking will eventually become necessary since it is highly unlikely that there will always be a service that offers the exact functionality the user requests. Based on these rankings, a user (or an agent that acts on behalf of user) can decide if he wants to make use of a Web service that does not match exactly the desired functionality or can use the service that match this functionality to some extent.

Matchmaking algorithms, presented so far, have been based on Service Profile, and have operated by comparing the required inputs and outputs against the advertised inputs and outputs. The intention of the thesis is to utilize the service model, based on the key concept of process, to enhance the service matchmaking. The process model does not just describe service in terms of inputs, outputs, preconditions, and effects, but decomposes the service into component subprocesses, and by doing this gives an insight into the manner how inputs are transformed into outputs. The DAML-S matchmaking algorithm, based on service process model and developed at the University of South Carolina [Ban02], has been modified and extended in this work to support semantic matchmaking of Web services written in OWL-S. The details about the algorithm functionality and its implementation are described further in the text.

For example, imagine an OWL-S advertisement for the location-aware content delivery Web service. As it is shown in Fig. 2.1, such a composite service is considered to be a sequence of processes corresponding to the retrieval of the geographical map that presents user's located area, choosing target location, determination of the content type user wants to receive on his mobile terminal (e.g. traffic or weather information), and subscribing to the selected content [TDJ$^+$05]. In the process of determination of target location where the selected content should be checked, the user can choose the target location either by selecting the landmark or by letting the service to determine his current location. Furthermore, when selecting the landmark, the user has the option to specify whether he wants to choose bookmarked landmark, mark the landmark using the map, or simply write the landmark name. Inputs expected from the user and outputs returned will be different in each case.

In case the user wants to select a landmark when using location-aware content delivery service, the traditional matchmaking based on the service profile would result with successful match only if the user provided inputs for all three possible options. However, by using the process model, the user has the ability to differen-

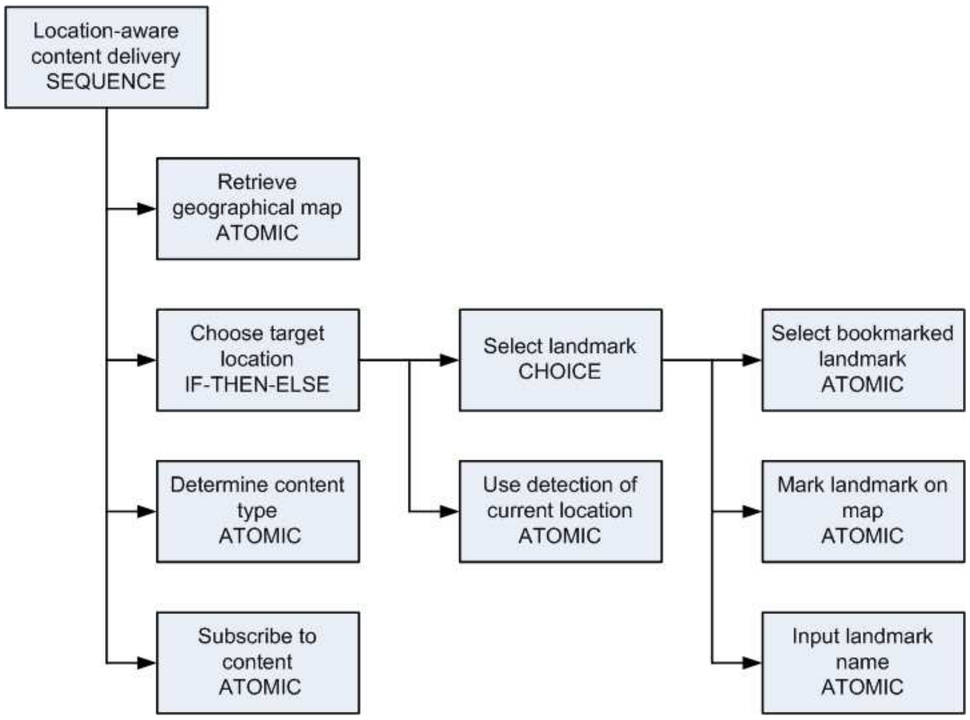

Location-aware content delivery SEQUENCE
Retrieve geographical map ATOMIC
Choose target location IF-THEN-ELSE
Select landmark CHOICE
Select bookmarked landmark ATOMIC
Determine content type ATOMIC
Use detection of current location ATOMIC
Mark landmark on map ATOMIC
Subscribe to content ATOMIC
Input landmark name ATOMIC

Figure 2.1: OWL-S model of the location-aware content delivery Web service

tiate among possible options and obtain a successful match by accepting an input corresponding to only one of these options.

The user can choose which parts of the OWL-S service description will use. The hierarchical decomposition of processes leads to the tree representation, where the root corresponds to the composite node representing the entire Web service and leaves correspond to atomic processes. As it can be seen in Fig. 2.1 the root node represents the entire location-aware content delivery service, and leaf nodes correspond to atomic processes of the service advertisement. Atomic nodes with corresponding inputs and outputs are listed in the Table 2.1.

Due to the better understanding of the algorithm, a matching degree is defined since it is not particularly useful to determine that an advertisement and the query are semantically incompatible. Based on the semantical equivalence of the query and the advertisement, following categories of matching degrees have been defined:

- *Exact.* The match is said to be exact when the requested service is the same as the advertised service;

- *PlugIn.* A plug-in match results when the requested service is subsumed by the advertised service. Hence, the advertised service can be substituted or plugged-in in place of the requested service. Such a match is less accurate but is also capable of satisfying the request;

- *Subsume.* When the requested service subsumes the advertised service, such a match is called Subsume. Consequently, the advertisement will only partially satisfy the request;
- *Fail.* The matchmaking results in failure if neither of the above matching degrees is satisfied.

These values have an ordering imposed on the following hierarchy: exact>plug-in>subsume>fail.

Table 2.1: Inputs and outputs for the location-aware Web service advertisement

Atomic Process	Inputs	Outputs
Retrieve geographical map		geographical map
Select bookmarked landmark	bookmarked landmark	landmark name
Mark landmark on map	location on map	landmark name
Input landmark name	landmark name	
Use detection of current location		current location
Determine content type	content type	
Subscribe to content		location-aware content

A sample query posted to the service advertisement can be constructed from specifications in Fig. 2.1 and Table 2.1. It consists of inputs provided by the user and expected outputs to be matched against inputs and outputs of the node, respectively. The query could, for example, request a service that accepts a bookmarked landmark and sends a location-aware content as the output. Taking into account the service process model illustrated in Fig. 2.1, the query results in a positive match even all the possible inputs were not specified, since it is possible to execute the service with the inputs provided and produce the desired output.

The implemented algorithm employs a tree data structure to represent the OWL-S Service Process Model advertisement and is recursive by nature. Each type of composite nodes, as well as atomic nodes have their own matchmaking algorithm. The matchmaking procedure starts by initiating matchmaking algorithm at the root node, which in turn invokes the matchmaking process for its child nodes, and so on until the process reaches leaf nodes of the advertisement tree.

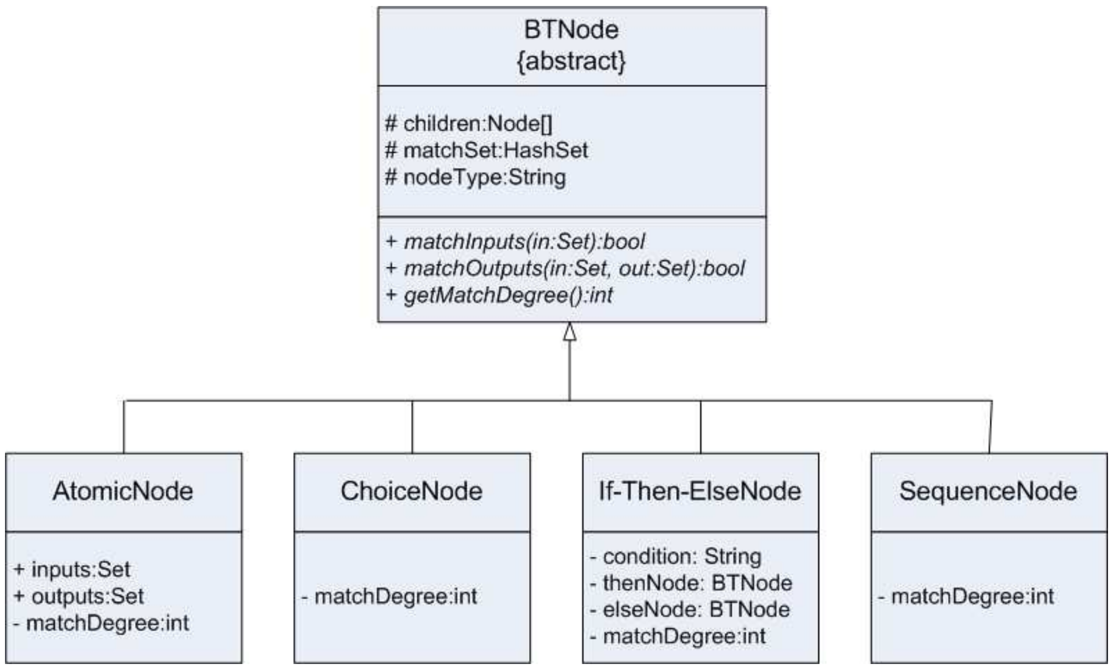


Figure 2.2: OWL-S nodes class diagram

The inheritance hierarchy of the OWL-S nodes is depicted in Fig. 2.2 with the UML class diagram. Each node type has its own algorithm for matching inputs and outputs.

BTNode represents an abstract class (binary tree node) that is extended by each node type, having the following data:

- *children*: an array of child nodes;
- *matchSet*: a set of outputs that are currently matched against the node;
- *inputs*: a list of inputs of the node;
- *outputs*: a list of outputs of the node.

Inputs and outputs are specified only for the Atomic Node, the node with no children. The *matchSet* of all the tree nodes is initially empty.

Split Node/Sequence Node

In the Split/Sequence node, inputs and outputs of the node are matched against query inputs and query outputs. In case any of children nodes fails to match on inputs, the entire sequence fails. Similarly, if desired outputs can be satisfied by all children collectively, the match is a success, otherwise a failure. Components of a split process need to be executed concurrently, while a sequence consists of subprocesses that have to be done in order.

The pseudocode for matching inputs of either a split or sequence node is the following:

```
Algorithm 2.2.1: MATCHINPUTS($In, numChildren, children$)

for $i \leftarrow 0$ to $numChildren$
  do if children(i) does not match In
  then { the entire sequence fails
       { return ( false )
return ( true )
```

In both algorithms (for matching inputs and outputs) *numChildren* denotes the number of the children nodes. Variable *children* represents the children of the sequence/split node, while *In* and *Out* are sets of inputs and outputs, respectively. Due to the code simplicity and redundancy, variables *numChildren* and *children* are written as arguments of both methods (matchInputs and matchOutputs), hence they are retrieved at the beginning of both methods.

The matchOutputs algorithm employs a recursion to find a distribution of the outputs over the children. It matches all possible distributions of outputs before returning a matching result.

The algorithm checks, at the beginning, if the list of outputs is empty and returns true to indicate that all outputs were matched successfully. Also, at the first call of the procedure, the *matching degree* is set to the maximum possible (*exact*).

Then, the list of outputs is iterated and its first element is placed in the *matchSet* of the first child node. The *matchOutputs* procedure is invoked recursively to match outputs in the matchSet against the first child.

If the match fails, the *output* is removed from the child's *matchSet*, and in the next iteration of the loop the output is matched against the next child of the node. If the *output* holds a negation of the value, meaning that an advertised service should not produce the desired output, it is removed from the other (already seen) children's *matchSet* as well. In that case an overall *matching degree* is set to *fails*, and *false* is returned as the result of matchmaking.

The outputs matching algorithm pseudocode is given here:

```
Algorithm 2.2.2: MATCHOUTPUTS($In, Out, numChildren, children$)

if $Out$ is empty
  then return ( true )
if $firstcall$
  then $matchDegree \leftarrow exact$
$firstOutput \leftarrow$ iterate $Out$
for $i \leftarrow 0$ to $numChildren$
  do
    $children(i).matchSet \leftarrow firstOutput$
    if $children(i)$ matchOutputs ($In$, $children(i).matchSet$)
      then
        if $matchDegree > children(i).matchDegree$
          then $matchDegree \leftarrow children(i).matchDegree$
        if $firstOutput$ is not negated or ($firstOutput$ is negated
         and $i = numChildren - 1$)
          then
            remove $firstOutput$ from $Out$
            $firstcall \leftarrow$ false
            if matchOutputs($In$, $Out$) and $In$ is match
              then return ( true )
      else
        remove $firstOutput$ from $children(i).matchSet$
        if $firstOutput$ is negated
          then
            for $j = 0$ to $i$
              do remove $firstOutput$ from $children(j).matchSet$
            $matchDegree \leftarrow fails$
            return ( false )
$matchDegree \leftarrow fails$
return ( false )
```

If the match succeeds, remaining outputs in the list are matched against the child's node through a recursive call to *matchOutputs* procedure. If the matching of remaining outputs against the node completes successfully, the *true* value is returned to indicate that all outputs from the current output onwards have been successfully matched. If, however, it is not possible to match remaining outputs against the entire node, the output is redrawn from the current's child *matchSet* and is tried to be matched against the next child in the sequence.

This process continues until the current output is matched against all children of the node. If no match is obtained for the current output or the remaining outputs of the list, a *false* value is returned as a result of the *matchOutputs* procedure.

Choice Node

A choice node consists of a set of processes from which one or more processes is selected to be executed. Inputs of this node are matched against query inputs. If it happens that any of children nodes satisfies inputs specified in the query, then the entire node matches inputs.

```
Algorithm 2.2.3: matchInputs($In, numChildren, children$)

for $i \leftarrow 0$ to $numChildren$
  do if children(i) matchInputs In
  then return ( true )
return ( false )
```

Outputs of the children of this node are matched against query outputs. If outputs for any child node match, then the match for the entire choice node succeeds. The highest match degree achieved among children nodes is set as the overall match degree. Otherwise, if none of child nodes matches, the match fails and a *false* value is returned as a result of the *matchOutputs* procedure.

```
Algorithm 2.2.4: matchOutputs($In, Out, numChildren, children$)

$matchDegree \leftarrow fails$
for $i \leftarrow 0$ to $numChildren$
  do if children(i) matchOutputs (In, Out)
  then {
    if $children(i).matchDegree > matchDegree$
      then $matchDegree \leftarrow children(i).matchDegree$
    return ( true )
  }
return ( false )
```

If-Then-Else Node

An If-Then-Else node selects one from two alternative processes for execution, depending on whether the associated *condition* is evaluated as *true* or *false*. Thus, two child nodes exist for the if-then-else node: the *thenNode* that corresponds to the process executed if the condition evaluates to *true*, and the *elseNode* that represents the process executed if the condition is set to *false*.

Inputs and outputs of this node are distributed over one of child nodes, depending on the evaluated condition.

Algorithm 2.2.5: MATCHINPUTS($In, condition, thenNode, elseNode$)

```
if condition = true
  then { if thenNode matchInputs (In)
           then return ( true )
  else { if elseNode matchInputs (In)
           then return ( true )
return ( false )
```

Algorithm 2.2.6: MATCHOUTPUTS($In, Out, condition, thenNode, elseNode$)

```
if condition = true
  then { if thenNode matchOutputs (In, Out)
           then { matchDegree ← thenNode.matchDegree
                  return ( true )
  else { if elseNode matchOutputs (In, Out)
           then { matchDegree ← elseNode.matchDegree
                  return ( true )
return ( false )
```

Atomic Node

An Atomic Node is, as its name says, an indivisible node - it has no subprocesses, is directly invocable, and can be executed in one step from the user's perspective.

Matching of inputs and outputs is performed as operation of iterating all inputs/outputs of one list and comparing them to members of the other list. The *inputs* denotes in the algorithm list of inputs of the node, while the *In* is the list of query inputs. Similarly, the *outputs* defines the list of node outputs and the *Out* represents the list of outputs specified in the query.

The matching result also takes into account negated inputs/outputs, that can be specified in the query as requirements for the requested service, that it should not request the specified inputs or produce the given outputs. Both algorithms are presented below:

Algorithm 2.2.7: MATCHINPUTS($In, inputs$)

```
if $inputs \subset In$
  then
    if $i \in inputs$ is not negated
      then return ( true )
      else return ( false )
  else
    if $i \in inputs$ is negated
      then return ( false )
      else return ( true )
```

The subset operation, defined in the algorithm, is borrowed from the math set terminology. It is used to determine the actual degree of match among two sets of outputs. That is, instead of returning only true or false based on exact matches, it uses the rule based engine to deduce if the match is exact, plug-in, subsumed, or there is no match possible.

Algorithm 2.2.8: MATCHOUTPUTS($In, Out, outputs$)

```
$matchDegree \leftarrow exact$
if $Out \subset outputs$ and $matchInputs(In)$
  then
    if $o \in Out$ is not negated
      then return ( true )
      else
        $matchDegree \leftarrow fails$
        return ( false )
  else
    if ∃ o' ∈ $Ontology$ (o' is a superclass of o ∈ $Out$) ⊂ $outputs$
      then
        if $o \in Out$ is not negated
          then
            if $matchDegree > plugIn$
              then $matchDegree \leftarrow plugIn$
            return ( true )
          else
            $matchDegree \leftarrow fails$
            return ( false )
      else
        if ∃ o' ∈ $Ontology$ (o' is a subclass of o ∈ $Out$) ⊂ $outputs$
          then
            if $o \in Out$ is not negated
              then
                if $matchDegree > subsumes$
                  then $matchDegree \leftarrow subsumes$
                return ( true )
              else
                $matchDegree \leftarrow fails$
                return ( false )
```

The algorithm sets the highest matching degree (*exact*) at the beginning. It first checks if the inputs and outputs specified in the query are contained in the service advertisement. Then, it checks for each output from the query if it is negated, and if true, returns *false* and sets the matching degree to the lowest one (*fail*). Otherwise, it returns *true*.

If it happens that the output specified in the query is not exactly the same as the one contained in the advertisement, the algorithm checks if this output is perhaps a subclass of any of the outputs specified in the advertisement. If this is the case (and if the output is not negated), it reduces the match degree to *plugIn*, if the present match degree holds a higher value. Hence, if the output is negated, the match fails, the match degree is set to *fail*, and *false* is returned as the result of the procedure.

To determine the class hierarchy and its relations, there must exist an OWL ontology, and it is specified in the algorithm as an external variable, called *Ontology*.

If the algorithm determines that the output specified in the query subsumes the output contained in the advertised service, it sets the match degree to *subsumes*, if the current match degree is higher than this one. In case of negated output, match process fails as it is described before.

Semantic agent for matchmaking of OWL-S service descriptions

The semantic agent that performs complex reasoning tasks, including interpreting service provider advertisements and user queries, is equipped with two components: OWL Inference Engine and OWL-S Matchmaker. The OWL Inference Engine component is used to transform OWL files (requested and advertised service) in the form appropriate for the OWL-S Matchmaker component, that using the matchmaking algorithm semantically compares transformed OWL files and calculates the degree of similarity between Web services.

In the context of e-market the semantic agent is called a broker agent (or a middle agent), since it plays a brokerage role between the buyer agents, wishing to purchase an access to a service that meets their needs, and the seller agents representing the service providers, offering similar, but not the same services. Thus, broker agents assist buyer agents in matchmaking of services, offered by one or more seller agents in the e-marketplace.

Figure 2.3 shows matchmaking process in the e-market as well as broker components used for matchmaking OWL-S service descriptions. In the matchmaking

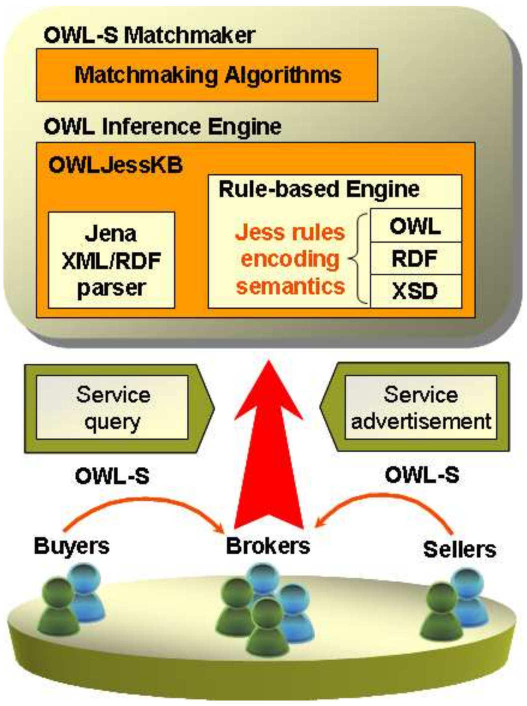


Figure 2.3: Broker agent components

process a buyer agent (which acts on behalf of a user) sends a query to the broker that compares the advertisements sent by seller agents (which act on behalf of service providers) and chooses the most appropriate service according to matchmaking algorithms.

The matchmaking process requires two different reasoning tasks: to abstract from query provider's required capabilities and to compare and match these capabilities with what available providers can really do. In order to accomplish these tasks the broker first uses OWL Inference Engine that, actually, represents OWLJessKB [owl] off-the-shelf component for reasoning with OWL ontologies. Brokers invoke OWLJessKB methods to load RDF as well as OWL documents. OWLJessKB uses Another RDF Parser (ARP), which is part of the Jena toolkit [Jen], to parse RDF/OWL documents. The OWLJessKB component is used to convert the OWL advertisement file into a set of Subject-Verb-Object (SVO) triples. These triples are then inserted into the Jess (Java Expert System Shell) [EH03] knowledge base and the rules of the OWL language are then applied by Jess. Jess is a rule-based engine used to create an agent knowledge base and populate it with facts and rules used to deduce new information. The broker agent then queries Jess to obtain the information necessary for building the advertisement tree. Finally, the request query is parsed and the matchmaking process is performed by OWL-S Matchmaker on the basis of previously described matchmaking algorithms.

The semantic (broker) agent that performs a matchmaking algorithm is implemented in JADE agent platform, that is briefly explained in the following section.

2.3 JADE agent platform

JADE (Java Agent DEvelopment Framework) [jad] is a leading open source FIPA (Foundation for Intelligent Platform Agents) [fip] compliant agent platform. As a consequence, JADE agents can communicate with other agents that comply with the same standard. FIPA is an international non-profit association of companies and organizations sharing the goal and the effort to produce standard specifications for agent technology. JADE represents a Java framework for development of multi-agent systems. The goal of the JADE is to simplify development of interoperable intelligent multi-agent systems while ensuring standard compliance through a comprehensive set of system services and agents.

JADE project has started in July 1998 as a joint development of Telecom Italia Lab and Parma University. Currently, JADE development is driven by "JADE board" composed of five industrial partners: TILAB, Motorola, Whitestein Technologies AG, Profactor GmbH, and France Telecom R&D.

The organizational structure of JADE is illustrated at Fig. 2.4 Basically, JADE can be run on a single machine or split over several hosts (physical machines). However, acting as a truly distributed system, JADE appears to the outside world as a single entity. A JADE system consists of several Agent Container instances, each one of them running in a separate Java Virtual Machine, but not necessarily on different hosts. The set of all containers is called the *platform*. A single special *Main Container* must always be active in the platform and all other "normal" containers register with it on startup. The main container is first started in a platform and every other container has to be told where to find (host and port) their main container.

Basically, agents are implemented as Java threads and live inside agent containers that represents runtime environment for their execution. Concurrent tasks can still be performed by one agent, and JADE schedules these tasks in a more efficient way than the Java Virtual Machine does for threads.

As a FIPA-compliant agent platform, JADE consists of AMS (Agent Management System), DF (Directory Facilitator), and ACC (Agent Communication

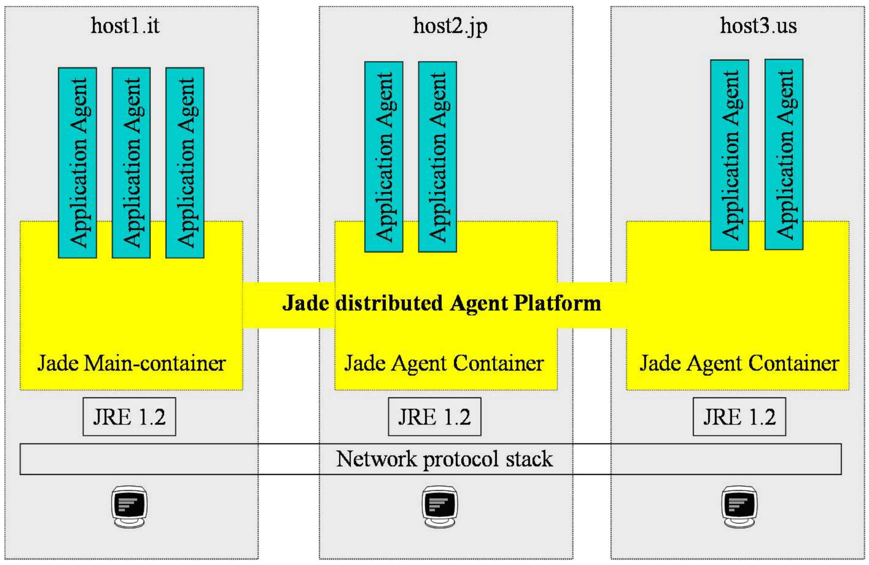


Figure 2.4: The software architecture of JADE Agent Platform

Channel). The AMS provides the naming service functionality, meaning that it ensures that each agent in the platform has a unique name, and it does so by maintaining a directory of agent identifiers (AIDs) and agent states. The agent gets a valid AID when it registers with the AMS. AMS also represents the authority in the platform that can create or kill agents on remote containers and there is always one AMS per platform. Directory Facilitator is an agent that provides Yellow Pages service by means of which an agent can find other agents providing the service he requires in order to achieve his goals. Agents wishing to advertise their services register with the DF. Visiting agents can then ask (search) the DF looking for agents which provide the services they desire. Agent Communication Channel is a Message Transport System that controls all messages within the platform, including messages to/from remote platforms. All three agents are automatically activated at the platform start-up.

Agents communicate by asynchronous message passing, where FIPA ACL is a language used to represent messages. Each agent has a sort of mailbox (the agent message queue) where the JADE runtime posts messages sent from other agents. Whenever a message is posted in the message queue, the receiving agent is notified. If and when the agent actually picks up the message from the message queue depends completely of programmer.

JADE agents are identified by a unique name and, provided they know each other's name, they can communicate transparently regardless of their actual loca-

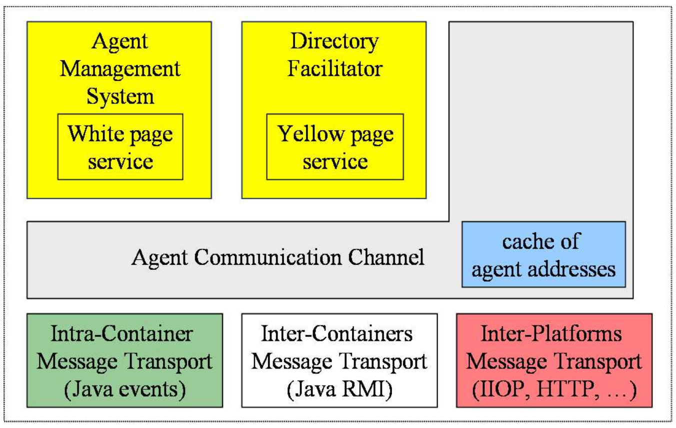

Agent Management System
White page service
Directory Facilitator
Yellow page service
Agent Communication Channel
cache of agent addresses
Intra-Container Message Transport (Java events)
Inter-Containers Message Transport (Java RMI)
Inter-Platforms Message Transport (IIOP, HTTP, ...)

Figure 2.5: The communication architecture of JADE

tion: same container, different containers in the same platform or different platforms (Fig. 2.5). In JADE, agents can migrate between connected containers.

The communication architecture of JADE offers flexible and efficient messaging by means of a number of protocols. Communication is possible via Java RMI, event-notification, IIOP, and HTTP protocol. Java events are used for effective and lightweight communication between agents that reside on the same host. Java RMI is used for communication between agents in different containers (intra-platform communication), while Corba (IIOP) and HTTP are utilized for communication of agents situated in different platforms (inter-platform communication).

2.3.1 LEAP add-on

LEAP [Cai05] is an extension of JADE designed to run on wireless devices such as mobile phones, PDAs, and Palm computers. It stands for Lightweight Extensible Agent Platform. The LEAP software was mainly developed within the scope of LEAP IST project and has been made available as a JADE add-on since JADE 3.0.

Due to the memory footprint of several MBytes, version 1.4 of Java platform (or higher) it requires, and wireless network constraints that are not present in the fixed network, JADE could not be run on small devices. The LEAP was created to solve these problems and when used with JADE, it replaces some parts of JADE kernel forming a modified runtime environment that can be deployed on a wide range of devices varying from servers to Java enabled mobile phones. It provides three modes of work to adapt to different circumstances (Fig. 2.6):

- *J2SE*: it can run in PCs and servers in the fixed network with jdk1.4 or superior;

- *J2ME CDC* configuration (former PersonalJava): it can run in handheld devices such as most of todays PDAs supporting J2ME CDC configuration;
- *J2ME CLDC/MIDP*: it can run in devices that support MIDP1.0 such as mobile phones.

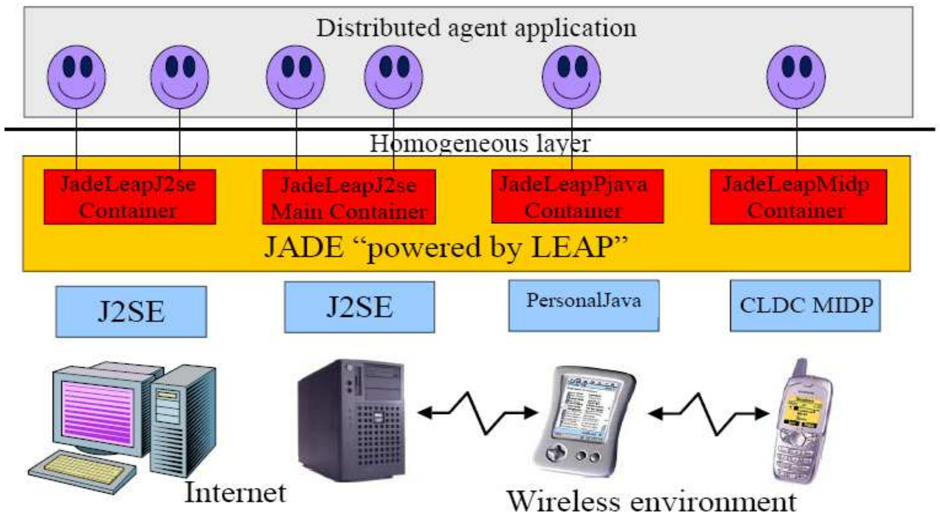


Figure 2.6: The LEAP execution environment

Though implemented differently, the three versions offer the same API to the developers, independently of the underlying type of network and supported java platform.

The JADE-LEAP runtime environment can be executed on handheld devices in two different ways (Fig. 2.7): the *"Stand-alone"* execution mode, where a complete container is executed on the handheld device, and the *"Split"* execution mode, where the container is split into a FrontEnd (actually running on the handheld device) and a BackEnd (running on a J2SE host) linked together through a permanent connection. Since the version 3.3, stand-alone mode is no longer maintained and tested, so its use is discouraged. The split execution mode is particularly suitable for wireless resource-constrained devices for the following reasons:

- The FrontEnd is more lightweight than a complete container;
- The bootstrap phase is faster;

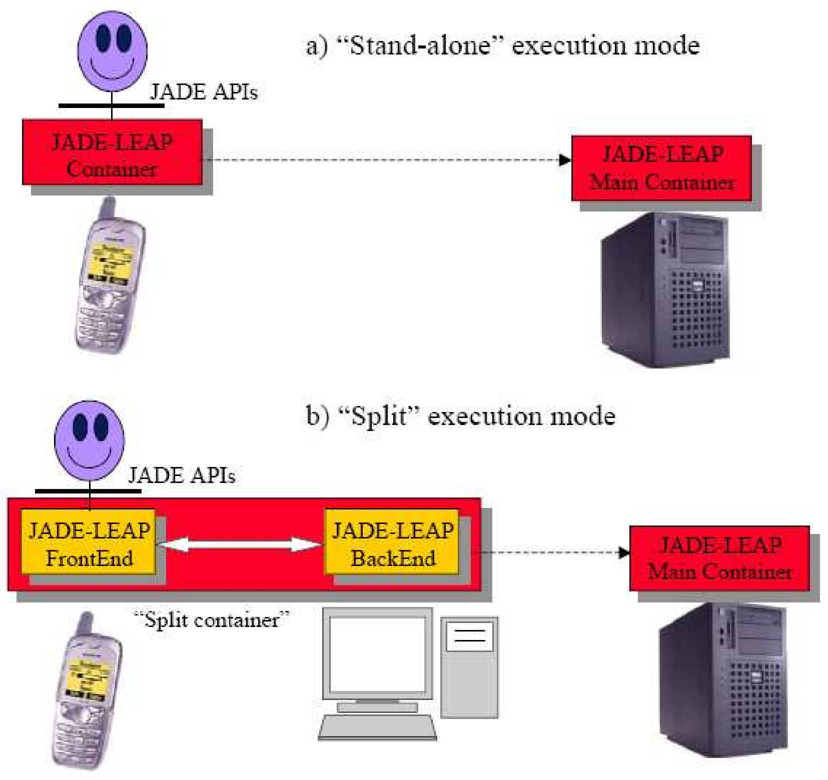


Figure 2.7: Execution modes

- Less bytes are transmitted over a wireless link.

Concerning a wireless connectivity between agents running in mobile devices and service-providing agents running in fixed PCs, JADE-LEAP works at a high communication level, establishing TCP/IP connections between containers and without caring about the physical means by which these connections are actually performed.

Chapter 3

Context-aware services

Context-aware computing is a computing paradigm in which applications can discover and take advantage of contextual information such as user location, time of the day, nearby people and devices, and user activity.

An idea of context-aware computing came from the growing number of wireless and mobile devices used in the user everyday life and the user's need to be served with timely delivered information anyplace, anytime and on any device, while on the move [DJ05].

Users want to access the same content and services on various devices, e.g. laptops, handheld devices, mobile phones, that have different processing capabilities, memory capacity, screen size, and support different media formats. They may also wish to change mobile device during the session and the service task is to recognize the user and enable him the smooth transition to the other device, while providing the continuity of the service use. This procedure is called personal mobility and is defined as the user ability to access other telecommunication services from any terminal and location, while on the move. Technical and service infrastructure (network, mobile positioning system, as well as available applications) can also change in the mobile environment.

The context of the mobile user should be continuously tracked and the so-called *context-awareness* is important for applications to be able to adapt their behavior according to the present situation and the conditions in the environment. The services that have implemented the context-awareness in their behavior are called *context-aware services.*

This chapter is organized as follows: Section 3.1 gives a definition and classification of context. The low-level context information, that is retrieved from the user environment, is used to create a higher-level context that can be used by context-aware applications. The context modelling and representation issues are discussed in Section 3.2. Location-aware services, that are represented as a subset of context-aware services, are described in Section 3.3. Section 3.3.1 gives a proposal of location semantics that can gather and structure different formats of user's location information, to build a higher-level context used by various location-aware services.

3.1 Definition of context

Mobile user context is defined [RLF00] as: *"any information that can be used to characterize the situation of the entity, where an entity can be a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves."*.

Schilit et al. [NAW94] claim that important aspects of the context answer the following questions *where are you, who are you with, and what resources are near you.*

Chen and Kotz [CK00] have extended the Schilit's division of context [NAW94] into four categories to achieve a better understanding of the concept:

- **Computing context**, including network connectivity, bandwidth, communication costs, and nearby resources such as printers, displays, and workstations;
- **User context**, described with user profiles, user location information, and nearby users and people;
- **Physical context**, referring to conditions in the environment, such as lighting, temperature, and humidity;
- **extended with Time context**, including time of the day, week, year, and the season of the year.

A definition of context-awareness is given in [BBC97] as: *a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.* Context-aware applications look at the who's, where's, when's, and what's (what the activities are occurring) of entities and use this context information to determine why a situation is occurring and how to act upon it (what action to perform) [tel03].

3.2 Context information

There are certain types of context information that are, in practice, more important than others. These are: *user location*, *identity*, *time*, and *activity*. They characterize the situation of a particular entity. These context types not only answer the questions of who, what, when, and where, but also act as indices into other sources of contextual information.

Context-aware services build on context information services, because operators of context-aware services are usually not able to create the complete required context on their own.

Context can be classified as *static* and *dynamic* [tel03]. Static context information describes invariant aspects, such as person's date of birth. The persistence of dynamic information can be highly variable, for example a user location that can change from minute to the next. The sensing of context changes is either a *continuous process*, in which the information requestor polls the information provider for new information sampled over fixed time intervals, or an *event-based mechanism*, that allows the information requestor to supply notification rules to the information provider.

Persistence characteristics influence the means by which context information must be gathered. The process of gathering context information from different context sources is called *context sensing*. Context information is usually provided in some required parameters and stored in the internal model. Typically, this refers to a *low-level context information*, such as user status, location, or time.

In a later stage, context information is refined, e.g. a location specified by a longitude and latitude pair is associated with a specific landmark name, address, or building, to denote a user's location of interest and it can be used in modelling *high-level context information* for context-aware services. Alternatively, this information can be aggregated with some other context information, for instance with time, to indicate that a user has reached the target location in certain time interval, meaning that his average speed was 80km/h and he probably travelled by car.

The problem with dynamic context is that information describing it can quickly become out of date. Also, context has multiple representations in different forms and at different levels of abstraction that need to interrelate.

Therefore, a representation of the context information should be applicable through the whole process of gathering, transferring, storing, and interpreting of context information. It is suggested in [HBS04] that the context representation should be:

- *Structured* - structured representation allows different forms of context information and provides the means for filtering relevant information;
- *Interchangeable* - context profiles must be interchangeable among the different components of the system, which requires a serializable representation;
- *Composable/decomposable* - by allowing decomposition and composition, profiles can be stored and maintained in a distributed way;
- *Uniform* - uniform representation of all information contained in context profile (device, user, additional context data) eases the interpretation during the process of service mediation and content adaptation;
- *Extensible* - profile representation format should allow future extensions;
- *Standardized* - there is a strong need for a standardized representation of the context information as the entities in the system may not belong to the same administrative domain.

These features of context information are essential when defining a context model approach. It is claimed that a commitment to an *ontology* [Gua98] is a first step towards implementing this model within a knowledge representation technology.

3.3 Location-aware services

User location has been one of the main drivers for building context-aware services. Modelling of location information is an interesting and importing topic, that influences the quality of developed location-aware systems. Location-aware services could be, therefore, seen as a subset of context-aware services, that adapt their behavior according to user location information.

For humans it is easy to understand and to exchange the location information, but for application it represents a quite difficult task. The complexity lies in the interrelation between different location formats and also in the different methods for location determination that can be utilized to produce these various forms of location information.

Nowadays, there are many positioning methods that can locate the mobile device and determine the mobile user's position, both in indoor and outdoor environments [pos01]. Positioning technologies are based either on technology in the network, or the technology in the headset. Each of them has implications in terms of the complexity, cost of implementation, the accuracy achieved, and future migration path, especially in terms of roaming. Almost all of them are based on the concept of triangulation, i.e. measuring multiple signals from different sources to give a reliable estimate of position. The source of signals may be in the wireless network or some other source, such as satellites.

The positioning methods overview is not intended to be a part of this thesis, they are just listed and will not be further explained: Cell ID or Cell Of Origin (COO), Angle Of Arrival (AOA), Time Of Arrival (TOA), Enhanced Observed Time Difference (E-OTD), and Assisted Global Positioning System (A-GPS).

3.3.1 Proposed location semantics for building higher-level context information

It is expected that in the near future most of devices will be equipped with the position sensing technology [DJ05], e.g. GPS, GSM, Bluetooth, WLAN, RFID. Therefore the adaptivity - the ability of various resources to interrelate under changeable connection conditions becomes an important issue in providing location-based services in mobile environments.

There are multiple resources of location information that can produce various forms of user location information, such as:

- geographic coordinates in the form of longitude, latitude pair using positioning system;
- WLAN/Bluetooth access point address in the indoor environment;
- MSISDN as the user terminal identification in mobile/fixed telecommunication network;
- IP address as the user trace in the local area network;
- URL as the homepage of the Internet site where the user was last registered.

Apart from the technology used, the location model should hide the process of location acquisition, providing higher level abstractions for application designers. In order to manipulate the given location information, the location model should provide the support for aggregation of various forms of location information, and assign them an appropriate user-understandable semantics. A term landmark is used in the thesis to assign a particular location name to a located user position.

The semantic interpretation of location should be structured in a generic presentation format and stored in the location repository database, from which it could be used as a foundation for building more comprehensive location-aware services.

A definition of location semantics is proposed in order to gather the specified interpretations of user location information. For instance, the following service could be considered: when the user accesses the Internet site of the Faculty of Electrical Engineering and Computing, his/her geographic location is automatically retrieved and he/she is offered with content and services available on that location:

```
<rdf:Description rdf:about="http://www.example.org/
                           UserProfile#Location">
    <ms:landmark>
        Faculty of Electrical Engineering and Computing
    </ms:landmark>
    <ms:homepage>http://www.fer.hr</ms:homepage>
    <ms:IPaddress>
        <rdf:Bag>
            <rdf:li>161.53.17.*</rdf:li>
            <rdf:li>161.53.19.*</rdf:li>
        </rdf:Bag>
    </ms:IPaddress>
    <ms:coordinates rdf:resource="http://www.example.org/
                           UserProfile#Coordinates"/>
    <ms:msisdn>+38516129999</ms:msisdn>
    <ms:wlanAP>192.168.2.19</ms:wlanAP>
    <ms:bluetoothAP>192.168.1.240</ms:bluetoothAP>
</rdf:Description>

<rdf:Description rdf:about="http://www.example.org/
                       UserProfile#Coordinates">
    <ms:longitude>N451083</ms:longitude>
    <ms:latitude>E153005</ms:latitude>
</rdf:Description>
```

This example shows how user location can be gathered from different sources and how its different formats can be structured in a way that a higher-level context, that is built upon it, can be further used in various location-aware services. The location semantics is serialized in RDF, a technique used to represent the knowledge using metadata, briefly described in chapter 1.

Chapter 4

Location-aware content delivery system architecture

The appearance of third generation mobile systems with the increased bandwidth and network coverage has caused a growing need for applications that will utilize the provided network resources. The new types of services, such as multimedia services, are currently being introduced in the network infrastructure by network operators: multimedia streaming services such as video telephony and audio streaming, as well as multimedia content engineering applications will play a significant role in the near future [eur01].

In the recent time information services gained a lot of popularity among mobile users, especially business type of people. The access to the right information at the right time and the right place is becoming a main driver to the development of new generation information services. Weather and traffic notification services, as well as stocks reports are examples of applications that rely on content dissemination. Personalization will be essential to services that deliver content to mobile users: content must be actively tailored to individuals based on rich collected knowledge about their preferences.

Mobile positioning has enabled a new set of services that adapt their behavior to the obtained user location information, called location-based services (LBS). Location-based information services are services that provide different information set to the user as he moves and changes the location.

Location-aware content delivery service presents a *mobile location-based information service* that enables the delivery of personalized content to mobile users, depending on user's current location, presence information, utilized terminal, and preferences. Presence information is introduced to describe user's current state and activity in order to assist the service in providing relevant content to the user and presenting it on the target terminal according to his current communication status. The presence attributes are reused from the Wireless Village Initiative [wvi02] and contain the information about user terminal status *(busy/idle/detached)*, his current location information, and information about his availability for communication *(available/not available/discreet)*.

Location-based publish/subscribe middleware offers means for content personalization: subscribers define characteristics of content that is of interest to them and the location where they would like to receive it, and get notified when such content becomes available. Subscribers can choose to subscribe to the specific content that is published on their current location, or to receive the specified content on some other location of interest (defined landmark). For the purpose of this thesis, location-based subscription is defined as a name of the topic used to publish content to interested parties, bound to the name of location where user would like to receive the notification. Locations are mapped to specific geographic coordinates, and referred to as the user *current location* or the defined *landmark*. Therefore, two types of location-based subscriptions are introduced and used in the thesis: *current location-based* and *landmark-based subscription*.

Nowadays, user can apply various mobile terminals to access the same content and services. The usage requirement of the service is to support user mobility, i.e. to maintain the same user identity irrespective of the terminal used and its network point of attachment. Terminals used may be of different types. They range from laptops, tablet PCs, Portable Digital Assistants (PDAs) to cellular phones, having different processing capabilities, memory capacity, screen size, and supporting different media formats. The pervasive presence of heterogeneous mobile terminals needs for a middleware component, that provides optimal adaptation of content and alters the service behavior according to the preferences explicitly expressed by the user and the current state of the involved terminals (screen resolution, software characteristics, communication status). The latest standards are used for resources, terminal capabilities, and user preferences description, proposed by the World Wide Web Consortium (Resource Description Framework - RDF) [RDF03] and Wireless Application Forum (User Agent Profile - WAP UAProf) [UaP99]. Attributes about user location and presence information, as well as personal data and preferences are defined in a RDF scheme that describes a user profile. The scheme reuses presence attributes from Wireless Village Initiative, and terminal capabilities and user preferences attributes from WAP UAProf scheme.

This section lists and analyzes requirements of location-aware content dissemination service and illustrates usage scenarios of a system offering personalized content dissemination to mobile users depending on their current location, utilized terminal, and preferences.

The chapter is structured as follows: Section 4.1 investigates functional requirements and usage scenarios that have guided the design of the reference architecture which is presented in Section 4.2. The reference architecture has a layered structure that consists of a set of components. The identified components and interaction between them is described in Section 4.2.4. Section 4.3 discusses the service architecture design for 3G mobile networks, with the entrance of new business roles in the value-chain. The service implementation is described with an architecture and service interface in the section 4.4.

4.1 Requirements and Usage Scenarios

For the purpose of usage scenarios two people and their everyday's lives are simulated in the Zagreb city area (Fig. 4.1). These people differ in age, occupation, and daily habits, and it is envisioned that they follow the same routine every day: they leave from *home* to *work* in the morning, then in the afternoon on the way home they stop to do the *shopping*, afterwards they go to some sort of *recreation*, and eventually return home. During that time they walk, take the tram or drive in the car to reach the target destination. For each user there is a set of landmarks that he/she visits every day:

- Home_i - the place of living for the user i, (H_i on map);
- Work_i - the working place for the user i, (W_i);
- Shopping_i - the shopping destination for the user i, (S_i);
- Recreation_i - the place for recreation activities for the user i, (R_i).

Each landmark is assigned a determined user position in the form of geographic coordinate (longitude, latitude), so that the current position of the user can be tracked and seen on the map, and the names of landmarks can be further used in the system for specifying a landmark-based subscription.

A number of usage scenarios for location-aware content delivery services in mobile environment are described. At first, the simplest scenario is analyzed that

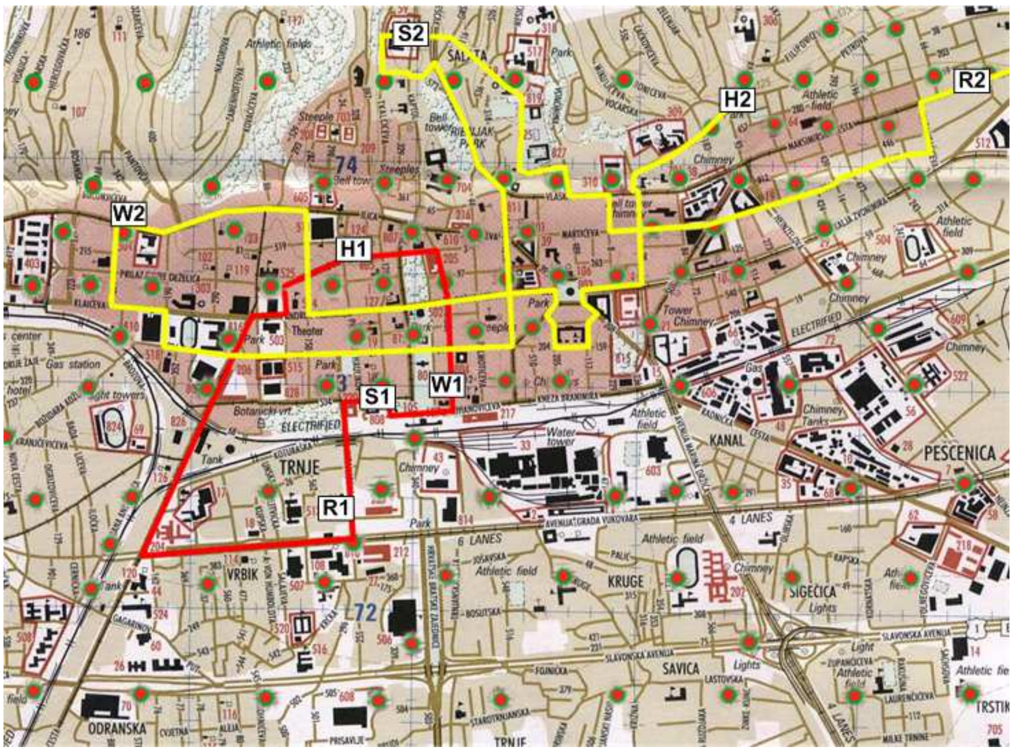


Figure 4.1: Zagreb city area

offers no support for location-aware content, and gradually is extended with more functionality for service users. In the first scenario a user employs a mobile phone *Sony Ericsson P900* to publish and receive the content. He chooses MMS as a preferred delivery method to receive the content on his mobile terminal. The second scenario enables the user to subscribe to the location-aware content and receive it on the current location. He can change the preferred delivery method to e-mail or SMS, or set to be unavailable and receive no content. In the third scenario a user can apply another mobile phone, *Nokia 6600*, to subscribe to the location-aware content and select the landmark where he would like to receive it. The user personal data, his applied terminal's capabilities, landmark-based subscription preferences, and presence information are stored in the profile repository at the operator infrastructure. This data is managed and retrieved through the user profile management subsystem. At the moment we will consider both components, the location-aware content delivery system and user profile management subsystem, as black boxes and discuss the functionalities from the user's perspective.

Scenario 1. User A wants to receive the *traffic notification service* that informs him about the current traffic conditions. He is just in the car on the way to his office. He accesses the service with his mobile phone, subscribes to the traffic news, and selects the MMS as a preferred delivery method. On the other side user B at the other part of the town finds himself in the middle of traffic congestion, takes a picture of the situation with the camera, and publishes the traffic report on the delivery system. The delivery system enables the publishers to define a topic for content classification and once the content is published, it initiates the delivery of a traffic report to all subscribers with a matching subscription. Before the delivery process begins, the system checks from the user profiles the subscribers' preferred delivery method and their availability status, and for each subscriber determines whether it will send the report to him and in what form (as SMS, MMS or e-mail). The traffic report is sent as MMS to the mobile terminal of the user A, and consequently A decides to take an alternative route to the office. If he needs an additional information, he can request a map of the located area with his current position drawn on it, to help him decide which direction to take.

In case the content cannot be delivered to the user's terminal (the user is currently set to be unavailable or he has switched off his mobile phone), it must be stored by the system for subsequent delivery. The system needs to provide a functionality to store the undelivered content until the conditions that enable the content delivery change (i.e. user becomes 'available' again or turns on his mobile phone) or until the validity period of the undelivered content expires.

Scenario 2. In this scenario user A would like to receive only traffic notifications that refer to his current location, i.e. that are published in his vicinity. He subscribes to the location-aware content with a current location-based subscription type. At some time during the day, user B passes near by the user A and publishes the traffic report. The system determines that the user B is located in the location area of user A and initiates the content delivery to the user A. In the meantime user A has changed his preferences, and set the preferred delivery method to e-mail. The system will read the user preferences stored in the user A's profile and check his availability status before it sends the content to his applied terminal. Consequently, traffic report will be delivered as e-mail to the user A's terminal.

Scenario 3. In this scenario user A would like to receive traffic notifications only in the morning when he is at home, and is preparing to go to work. He doesn't want to be disturbed with receiving traffic reports at any other time and location. Due to the low power supply condition, he switches to his other terminal, Nokia 6600 to access the service, and subscribes to the location-aware content with a landmark-based subscription type. The system recognizes the user with a different terminal and enables him the continuity of the service use. The user A selects *Home* as a desired landmark, and submits his subscription to the system. In any time when user B publishes the traffic information, the system will compare the user A's location with the specified landmark's position, and decide whether to start the delivery of the traffic report to the user A.

A *content-based filtering* is needed to provide a set of rules/filters to apply on the messages for the 'traffic' topic and deliver only the ones that match subscriber's preferences. The matching rule will be met when the publisher and subscriber are located in the same location area (in the case of current location-based subscription), or the subscriber has reached his landmark's position (landmark-based subscription) and the publication contains the same topic name as specified in the subscription. The former match condition is defined as a *location match*, and the latter as a *topic match*.

Due to the variations in user terminals' capabilities, *content adaptation and presentation* are important parts of all scenarios. For example, user A will receive a map of his location area that fits his applied terminal's screen. The content has to be adapted and displayed on terminals with different processing capabilities, screen size, and supported media formats. User profile comprises information about applied terminal's capabilities.

Service functional requirements can be specified after the scenarios discussion. They have guided the design of the system architecture:

- **User profile generation**: user profile should be generated for every user and his applied device, based on his personal data, terminal capabilities, and preferences to receive location-aware content based on preferred delivery method (SMS/MMS/e-mail), availability for communication, and name of the topic;

- **Landmark declaration**: user can declare his current position as a landmark giving it a semantic meaning (e.g. Home, Work, Shopping, Recreation);

- **Location-aware content**: user can create content related to his current location or to one of the specified landmarks, with a possibility to define a validity period for the contained information. The system should provide the user with a mechanism to disseminate this information to other interested users, and to retrieve the location-aware content (published by other users) on the targeted device, when he is located at the specified location;

- **Preview of most frequently visited landmarks**: user can view his most frequently visited landmarks;

- **View map**: user can retrieve a part of the city map displaying his current location area;

- **User preferences modification**: user can modify his initial settings stored in the profile: the availability status, the list of preferred contacts and priorities, and preferences to receive different information set at the specified location;

- **Terminal heterogeneity**: user can access the application from different terminals, and the system should recognize the user with a different terminal and enable him to continue with the service use.

4.2 Reference architecture

This section begins with a definition of the architecture and then proposes the reference architecture of the location-aware content delivery system based on the specified functional requirements.

Numerous definitions have been proposed describing the concept of the architecture, widely varying in the scope and emphasis. The main focus of the system architecture definition is on describing the structure of *components*, relationships between them, and the way they interact dynamically. The following definition is adopted in the thesis:

An architecture of the system is defined as the structure or structures of the system, which consist of elements and their externally visible properties, and the relationships among them. [CBB$^+$02]

Elements incorporate hardware and software components in the scope of the system's architecture.

A reference architecture refers to the system architecture that has already been created for a particular domain of interest. It typically includes many different architecture styles, applied in different areas of its structure.

In this section the reference architecture of the location-aware content delivery system is proposed (Fig. 4.2). The architecture is logically divided into four layers: application layer, service layer, communication layer, and transport layer, following the 3GPP/Open Service Access (OSA) approach [3gp00].

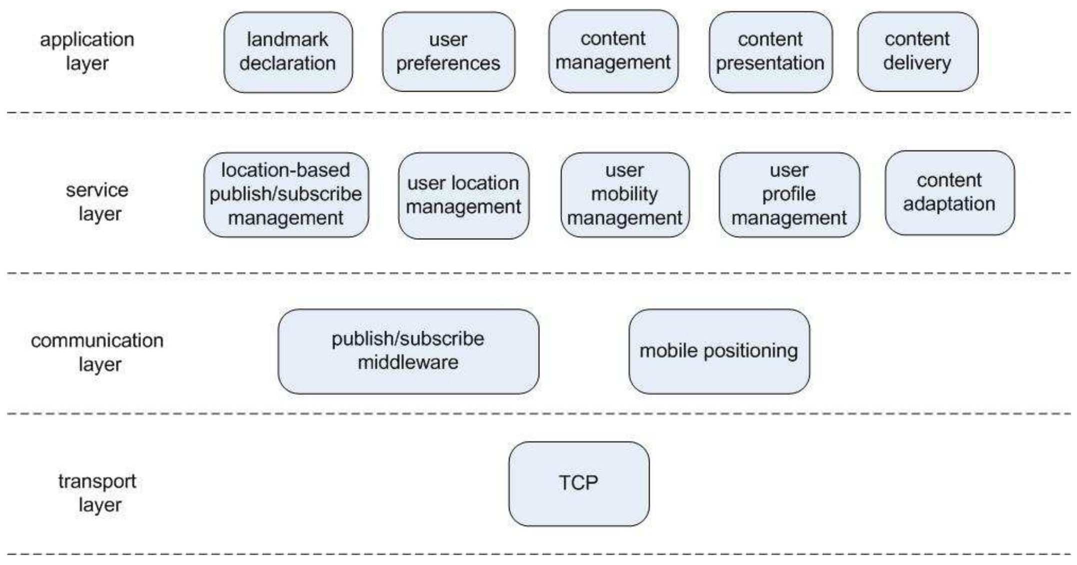


Figure 4.2: Reference architecture

The *application layer* enables subscribers to declare landmark(s) as their location(s) of interest, binds it to their current position at the moment they have been located on it, and uses in the subscription to specify where they would like to receive the desired content. It also provides the means for subscribers to modify their preferences stored in the user profile repository. Furthermore, it offers publishers possibility to create and manage location-aware content, and deals with content presentation and delivery to the subscriber's applied terminal.

The *service layer* contains services that are needed to provide location-aware content delivery to the subscriber's applied terminal. These services need to access the network elements (user profile, status, and location server) and to be suitable for various underlying networks and supported terminals. The services are: location-based publish/subscribe management, user location management, user mobility management, user profile management, and content adaptation.

The *communication layer* provides the publish/subscribe messaging functionalities and terminal positioning to the higher layers.

The identified services in the service layer are defined as generic services and use the communication and positioning capabilities from the underlying layer. Nowadays they would be deployed as Web services, that are managed and offered from a mobile operator infrastructure.

4.2.1 Communication layer

A publish/subscribe mechanism is used in the system as a means of communication between two types of users: publishers and subscribers [DP03]. Publishers define the content that is submitted to the service for the subsequent delivery to subscribers. Subscribers define subscriptions that describe the type of content they are interested in receiving. A notification produced by the publisher is delivered to the subscriber if the publication matches the subscription, according to the specified matching rule. The topic-based subscription scheme is used to match published events against subscriptions.

To offer support for mobility, the publish/subscribe middleware should provide temporary storage for published content for outreached subscribers using the store and forward mechanism for persistent notifications. It serves as a distribution media for notifications to subscribers' applied terminals.

The publish/subscribe mechanism is extended in the service layer to support location-based subscriptions and publications of location-based content. This enables publishers to publish the content related to their current geographical position, while subscribers declare interest in receiving the particular information set at their current location or at the specified landmark. The mobile positioning component is needed to locate the end-user's terminal and to determine its position. In order to manipulate the given location information, the appropriate user-understandable semantics is assigned to this information by a user location management component. The semantic interpretation of location is then used for location-based content filtering, i.e. comparing the locations associated to both publications and subscriptions and filtering the notification events to the matching condition.

4.2.2 Service layer

The location-based publish subscribe component serves as a mediator between the application layer and the publish/subscribe middleware. It manages and coordinates other services: Firstly, it activates and deactivates user subscriptions according to user availability status and user subscription preferences regarding location. Secondly, it provides location-based content filtering, i.e. notifications produced by publishers are filtered according to the location associated with publication and subscription events. Thirdly, it manages the content adaptation according to capabilities of the applied user's terminal, which is being tracked by a user mobility management component. The component cooperates with user location management, user mobility management, user profile management, and content adaptation component, and relies on publish/subscribe middleware and mobile positioning for receiving and disseminating the location-aware content to interested subscribers.

The *user location management* component binds landmark names specified by subscribers to their current location information.

The *user mobility management* component keeps track of subscriber's presence mode, which involves information about his applied terminal status and his availability for communication.

The *user profile management* component administers user profiles and enables subscribers to customize service behavior with the information stored in profile. The user profile is defined for every subscriber's applied terminal. The information in the profile comprises subscriber's personal data, terminal capabilities, presence information, and location-based subscription preferences.

A subscriber can declare which subscriptions apply to a particular landmark and how he would prefer to receive the content on the applied terminal (via SMS, MMS, or e-mail).

The *content adaptation* component adapts the content to fit the subscriber's applied terminal. For example, an image has to be transformed to the supported format and cropped to the actual screen size to be displayed on the phone screen.

4.2.3 Application layer

The application layer contains components that manage content delivery and presentation according to user preferences. It is affected by a type of the content that is distributed to subscribers and the particular application purpose. The user specifies his requirements and manipulates the service behavior through these components. The *landmark declaration* component retrieves the subscriber's current position and enables the subscriber to assign the name of the landmark to this position. The *user preferences* component offers the subscriber the possibility to modify his preferences stored in the profile, which are the following: subscription preferences, preferred delivery method, and availability status. The *content management* component enables the publisher to select the topic and create a content which will be published on the specified topic. The publisher can decide whether this content will be location-based or not, and in the case of location-based content the component will retrieve the publisher's current position and bind it to the publication. He can also specify the validity period of this information set, after which it will not be distributed to subscribers. The *content presentation* component is responsible for terminal-dependent content representation: the content must be adapted to suit the applied terminal screen size and must be transformed to one of terminal's supported formats. The java platform for constrained mobile devices (J2ME CLDC/MIDP) takes care of content presentation and structuring on the low-level user interface [Knu03]. The *content delivery* component delivers the content to the subscriber's applied terminal using the preferred delivery method specified in the user profile.

4.2.4 Component interaction

The interaction between components of the proposed reference architecture is presented using Unified Modelling Language (UML) sequence diagrams [BJR98]. The following use cases are described: process of user's subscribing to the topic with current location-based subscription, landmark-based subscription, and non location-based subscription, as well as content publishing and content delivery.

The sequence diagram (Fig. 4.3) depicts the component interaction for two representative use cases: current location-based subscription (a subscriber subscribes to the topic with a current location-based subscription) and location-aware content publishing (a publisher publishes a content that is related to his current location). The subscriber sends a subscription request from *GUI* to *Location-based Pub/Sub Mng* component. The *User Mobility Mng* component recognizes the user and his terminal with the given msisdn. After the received subscription request, the *Location-based Pub/Sub Mng* component sends the request to *Mobile positioning* component to locate the subscriber, and when received location result, it creates a current location-based subscription and subscribes to the topic at the *Pub/Sub* component. When a publisher at some time publishes the content, the *Location-based Pub/Sub Mng* component will receive a notification. The *Mobile positioning* component periodically sends a location push to the subscriber, containing his new geographic position. If it determines that the publisher and subscriber are located at the same position, it will deliver the content to the subscriber. The content is previously adapted to subscriber's applied terminal's capabilities by the *Content adaptation* component. A subscriber receives the adapted content on his mobile terminal via a preferred delivery method (SMS, MMS or e-mail) if his status is set to available.

The sequence diagram in Fig. 4.4 presents a scenario where a subscriber sub-

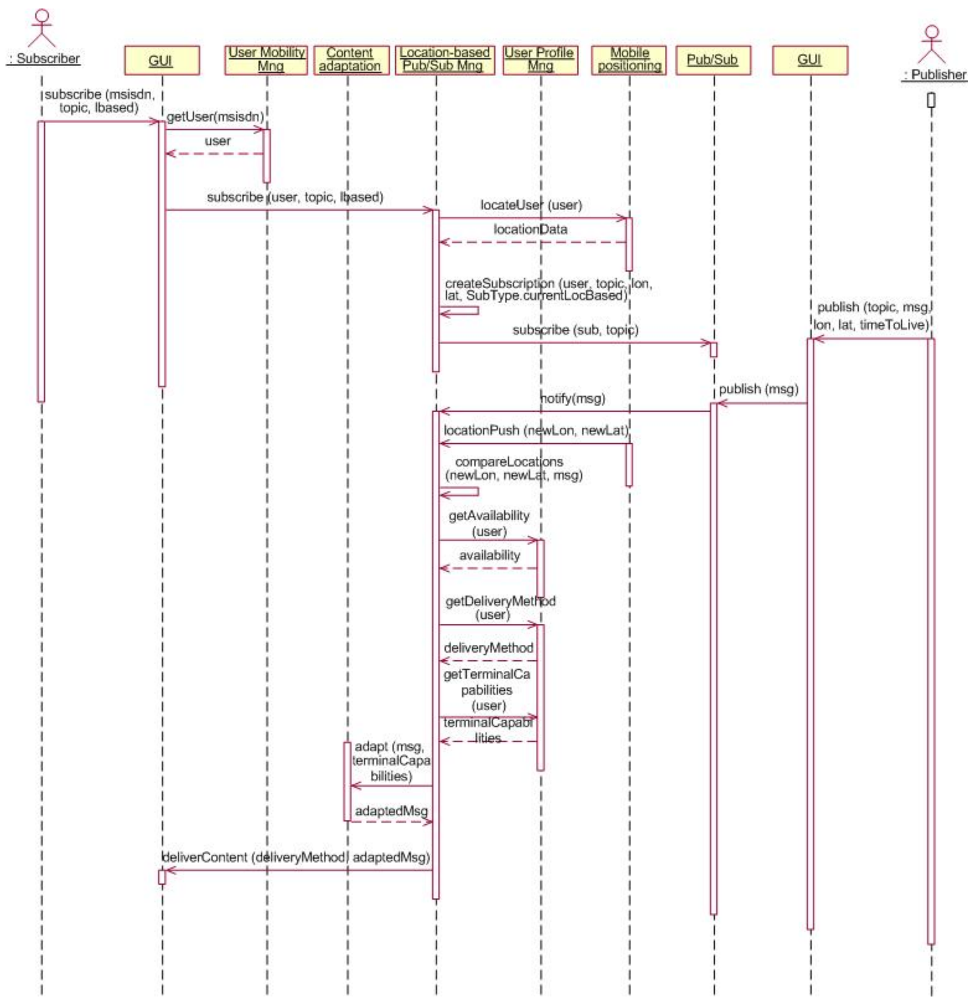

: Subscriber
GUI
User Mobility Mng
Content adaptation
Location-based Pub/Sub Mng
User Profile Mng
Mobile positioning
Pub/Sub
GUI
: Publisher
subscribe (msisdn, topic, lbased)
getUser(msisdn)
user
subscribe (user, topic, lbased)
locateUser (user)
locationData
createSubscription (user, topic, lon, lat, SubType.currentLocBased)
subscribe (sub, topic)
publish (topic, msg, lon, lat, timeToLive)
publish (msg)
notify(msg)
locationPush (newLon, newLat)
compareLocations (newLon, newLat, msg)
getAvailability (user)
availability
getDeliveryMethod (user)
deliveryMethod
getTerminalCapabilities (user)
terminalCapabilities
adapt (msg, terminalCapabilities)
adaptedMsg
deliverContent (deliveryMethod, adaptedMsg)

Figure 4.3: Current location-based subscription

scribes to the topic with a landmark-based subscription. It is assumed that the subscriber has previously defined his landmarks and that they are stored in the repository. The subscriber subscribes to the topic with landmark-based subscription, similarly to the previous scenario. The difference lies in the way how the *Location-based Pub/Sub Mng* component determines whether the subscriber has reached the target location. It compares the location of the subscriber received by location push with the location of the landmark specified in his subscription. If the location match condition is met, the *Location-based Pub/Sub Mng* component delivers the (adapted) content to the subscriber.

The sequence diagram in Fig. 4.5 shows the simplest scenario when a subscriber subscribes to the topic with a non location-based subscription. In this case the *Mobile Positioning* component is not involved in the component interaction.

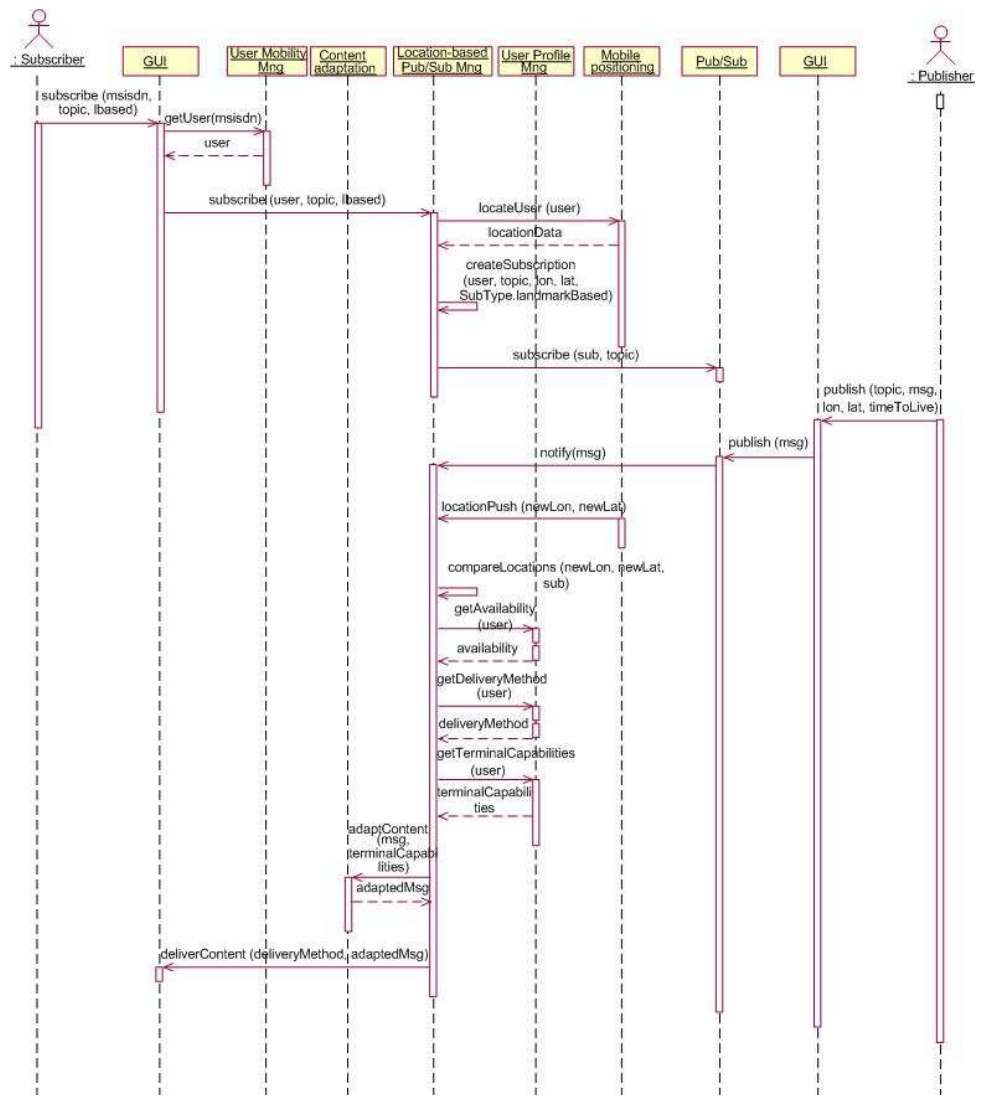

: Subscriber
GUI
User Mobility Mng
Content adaptation
Location-based Pub/Sub Mng
User Profile Mng
Mobile positioning
Pub/Sub
GUI
: Publisher
subscribe (msisdn, topic, lbased)
getUser(msisdn)
user
subscribe (user, topic, lbased)
locateUser (user)
locationData
createSubscription (user, topic, lon, lat, SubType.landmarkBased)
subscribe (sub, topic)
publish (topic, msg, lon, lat, timeToLive)
publish (msg)
notify(msg)
locationPush (newLon, newLat)
compareLocations (newLon, newLat, sub)
getAvailability (user)
availability
getDeliveryMethod (user)
deliveryMethod
getTerminalCapabilities (user)
terminalCapabilities
adaptContent (msg, terminalCapabilities)
adaptedMsg
deliverContent (deliveryMethod, adaptedMsg)

Figure 4.4: Landmark-based subscription

The subscriber subscribes to the topic, and at some time when the publisher publishes the content on that topic, the adapted content is immediately sent to the subscriber's applied terminal.

4.3 System architecture

The system architecture (Fig. 4.6) consists of components from the proposed reference architecture. It is built around two distributed components: the location-aware content delivery service and the user profile management service.

The *user profile management service* provides the interface for the user equip-

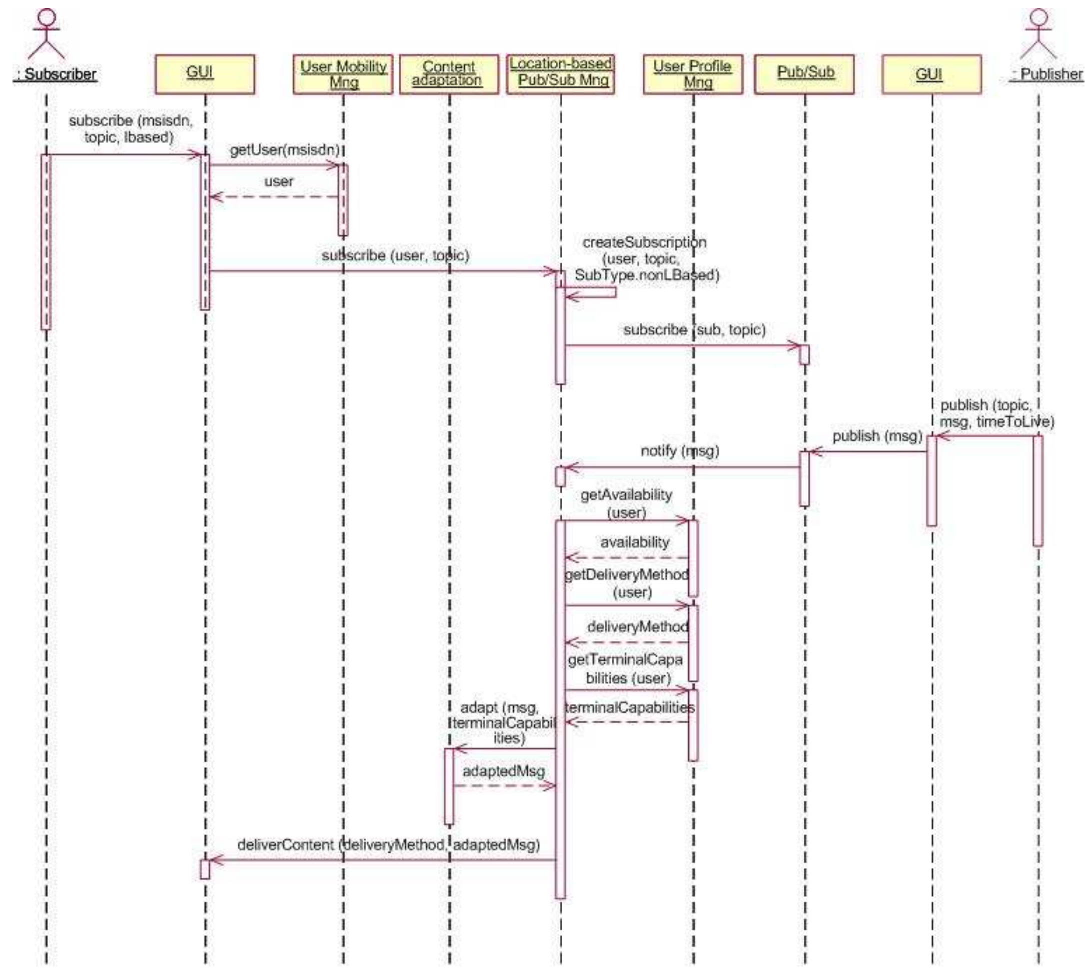


Figure 4.5: Non location-based subscription

ment and other system components (through the location-aware content delivery service), to access and modify information stored in the profile and location repository. On the other hand, it maintains up-to-date records with time-sensitive information and provides the functionality of creating and administrating user profiles and user defined landmarks in the profile and location repository, respectively.

The *location-aware content delivery service* is responsible for receiving HTTP requests from the user equipment, routing them to one of the following component handlers: positioning handler, map handler, content provider handler, location handler, status handler, and profile handler, that will process the request to the target component and return the received result to the user. An implementation of the location-aware content delivery service and user profile management service is briefly described in the following section.

Two actors that interact with the system are identified: an administrator and a

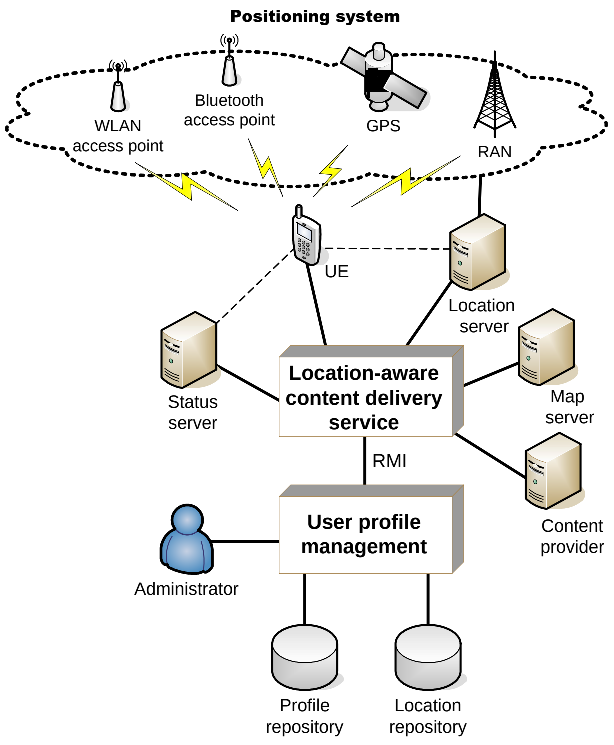


Figure 4.6: System architecture

user. The administrator can access the user profile management service through the administrator interface, and his responsibility is to administrate user profiles. The user can access the application using the user equipment (UE), that communicates with other system components through the location-aware content delivery service.

The user equipment (UE) can be tracked from various positioning resources, such as WLAN and Bluetooth access points in indoor, as well as from GPS and Radio Access Networks (RANs) in outdoor environments (Fig. 4.6). Depending on the utilized positioning method, the application installed on the UE can be designed to periodically report its position to the *location server* (if the positioning is terminal-based), or the positioning system can be configured to periodically report the user position to the location server (in the case of network-based positioning). The system should enable the user to find out his own location as well as the location of other entities by initiating the request to the location server. It is assumed that in the worst case scenario at least one system should function properly. Different forms of location information generated from positioning systems are collected and integrated into the generic presentation format and stored with the user-defined landmark name in the location repository.

The *status server* tracks changes in the user equipment status (busy/idle/detached) and reports this information on request to the location-based publish/subscribe service. User can set/modify his availability status anytime using the client application, executing on the UE. Should any information become available (from the UE or the location server), the system will update the user profile with this information in profile repository.

The *map server* loads the map of the city and dynamically crops the region centered around the user position with dimensions of user terminal's screen.

The *content provider* is presented by a publish/subscribe component that receives users' subscriptions for content and delivers it to them when the appropriate information set is published. The delivery method is specified for every user's applied terminal in the user profile. Subscriptions can be location-based or non-location-based. If the subscription is location-based, the component will first check if there are users located in the specified area, and if true, deliver the content.

4.3.1 Service architecture for 3G mobile networks

Considering the scenario for service provisioning in 3G mobile networks, it is necessary to change the service model from the "monolithic" 2G telecommunication model, where a mobile operator has played a central role in the value chain, and include several more actors in the provisioning of mobile services (like connectivity provider, service component provider, broker for 3rd party service access, portal content provider, and service/application provider) [BCMS]. The *connectivity provider* should enable the mobile operator a functionality to configure and customize the connections to provide flexible value-added service offerings (e.g. different QoS policies). The *service component provider* should enable the mobile operator to provide in a secured, controlled, and accountable way a customized and extensible set of service components to be used by 3rd party application providers. The *broker for (3rd parties) service access* should enable the mobile operator to provide its customers the personalized, secured, and authenticated access to a wide set of affiliated service providers. The *portal content provider* should enable the mobile operator to provide different content offerings to its customers from any point of origin. The *service/application provider* should enable the mobile operator a support to develop its own services and possibly improve development process.

The reference architecture should, for these reasons, be mapped to a new service architecture model designed for third generation mobile networks. The architecture must be considered from an end-to-end perspective, including end-users' terminals, application servers, but mainly focused on components providing telecommunication features in the mobile operator's infrastructure.

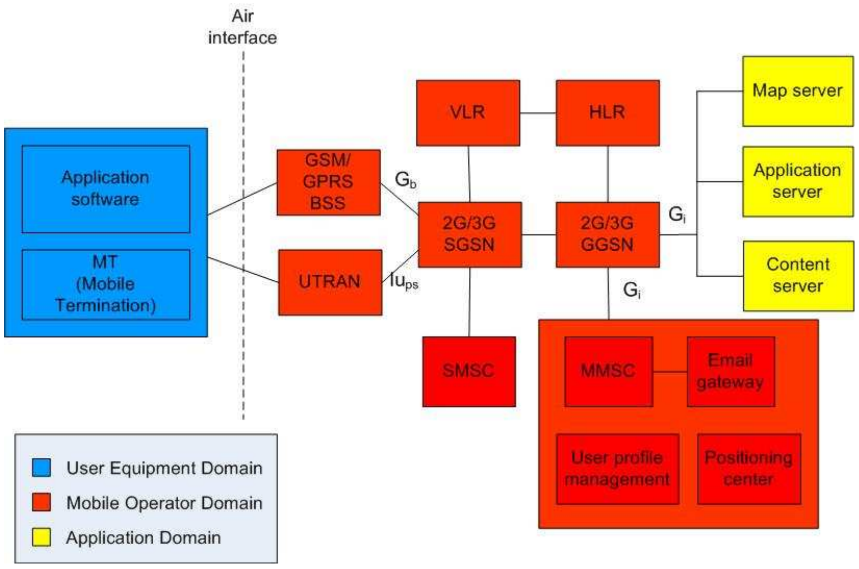


Figure 4.7: 3G service architecture

The service architecture (Fig. 4.7) consists of three different components:

- **User Equipment environment**: provides the users capabilities and functions to access the service with any terminal implementing this component, e.g. mobile phones, PDAs, smart phones, PCs with PCMCIA card for mobile network connection, etc. The terminal has to be equipped with the application software in order to access the service;

- **Application environment**: provides the service and portal content providers capabilities and functions to offer end-users value-added services and content. Systems that implement this component are Web servers, application servers, and telecommunication platforms (e.g. Service Nodes, publish/subscribe system);

- **Mobile Operator environment**: provides several capabilities and functions to handle the connectivity among terminals or between terminals and application environments. It provides, in addition to the basic connectivity, more advanced capabilities, such as: terminal positioning, session control, user profile management, user authentication/authorization, messaging store/forward, etc. The systems that implement this component are

GSM/GPRS BSS, UTRAN, SGSN, GGSN, VLR, HLR, SMSC, MMSC, Positioning center, E-mail gateway, and User profile management component.

The business roles introduced in the functional description of the service architecture are the following: a mobile network operator that manages the mobile operator environment, and possibly the application environment, in case it wants to play the role of the (internal) service provider; a user that manages and interacts with the user equipment environment; and service providers and portal content providers that manage the application environment.

4.3.2 Location-based publish/subscribe web service

Having the service architecture in mind, the location-based publish/subscribe service will be hosted by the service provider, who will enable the mobile operator to provide its functionality to the customers, independently of their point of attachment.

Web services are designed to provide interoperability between diverse applications [TDJ+04]. Platform and language independent interfaces of Web services allow easy integration of heterogeneous systems. Because of this flexibility, the idea of mobile Web services is to integrate datacom services with mobile applications, as well as mobile telecom services with PC-based applications.

Architecture

Figure 4.8 depicts the architecture of the proposed location-based publish/subscribe web service. The service is designed to be a Web service, used by mobile application, which implements the service layer of the proposed reference architecture and utilizes functionalities of publish/subscribe middleware placed in the communication layer.

The layered approach is adopted because other applications, such as content delivery, can be built upon the location-based publish/subscribe mechanism and they all rely on specific data transport components, such as e-mail, SMS, and MMS component. The publish/subscribe middleware in the communication layer is realized using Joram, an open source JMS API [jor], and is extended in the service layer to support location-based subscriptions and publications of location-aware content.

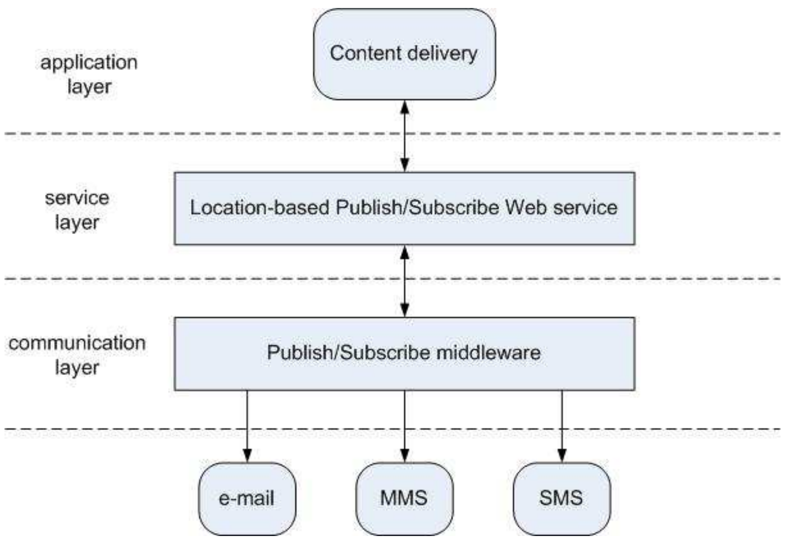


Figure 4.8: Location-based publish/subscribe web service

Service interface

The location-based publish/subscribe web service offers the remote service interface to the mobile users, which consists of the methods: publish, subscribe, and unsubscribe, as illustrated in Fig. 4.9 [DJ05].

A publication represents an event distributed at the publisher's current location. It conveys the following information: name of the topic, serialized object containing content elements (text, image), content validity period, and location coordinate (longitude, latitude). Users that are enrolled in the service as publishers use the *publish()* method to create and distribute the publication.

Regarding the type, the publication can be *location-based*, when it is bound to publisher's particular location coordinate, or *non-location-based*. Considering the validity period of the information set, the publication can be *persistent* or *non-persistent*. Non-persistent publications are disseminated *at the time of the publication* using the two *publish()* method variants leaving out *timeToLive* argument. With the persistent publication, an event can be distributed to the interested subscribers located at the determined location *before the event's validity period is expired*. Persistent publications are performed using other two *publish()* variants taking *timeToLive* argument.

A subscription is a request to receive a notification from the publisher when the publication matches the subscription event, according to the following matching rule: the matching rule condition is met when the publisher and the subscriber

```
 public interface LBPS extends Remote {

    void publish(
        String topic, Message message
    ) throws RemoteException;
    void publish (
        String topic, Message message, String longitude, String latitude
        ) throws RemoteException;
    void publish (
        String topic, Message message, long timeToLive
        ) throws RemoteException;
    void publish (
        String topic, Message message, String longitude, String latitude,
        long timeToLive) throws RemoteException;
    Subscription subscribe (
        User user, boolean onCurrentLocation, String topic
        ) throws RemoteException;
    Subscription subscribe (
        User user, String topic, String longitude, String latitude
        ) throws RemoteException;
    void unsubscribe (
        Subscription sub
        ) throws RemoteException;
}
```

Figure 4.9: LBPS interface

are both located at the same position (in the case of *current location-based subscription*) or the subscriber has reached his landmark's position (*landmark-based subscription*), and the publication contains the same topic as specified in the subscription. The former match condition is defined as a *location match*, and the latter as a *topic match*.

The subscription is performed using the *subscribe()* method taking four arguments, i.e., (1) user, (2) topic, (3) longitude, and (4) latitude, if the subscription is *landmark-based* (where longitude and latitude correspond to subscriber's landmark coordinate pair). Alternatively, the *subscribe()* variant with three arguments, i.e., (1) user, (2) onCurrentLocation, and (3) topic, can be used to denote that the subscription should be location-based or not. In case of location-based subscription the subscriber's current position will be used. The *subscribe* method returns a subscription object, which can be used by a subscriber to cancel the corresponding subscription with the *unsubscribe()* method.

4.3.3 User Profile Management Web service

User profile management is hosted by the mobile operator, providing the remote interface to the user equipment domain and the application environment to access and modify the information stored in the profile and location repository. On the other hand, it maintains up-to-date records with time-sensitive information and provides the functionality of creating and administrating user profiles, as well as user defined landmarks in the profile and location repository, respectively. User can modify his preferences and define his landmarks using the client application (installed on his mobile terminal), which cooperates with the user profile management service.

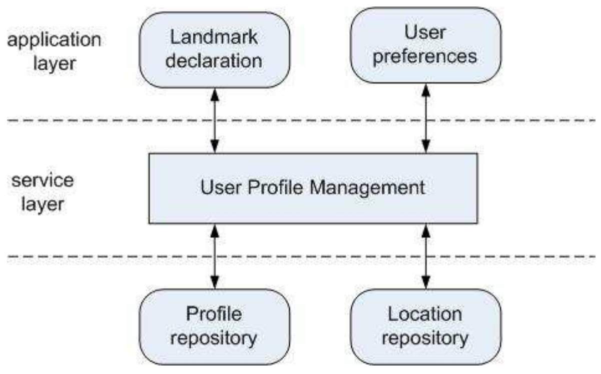


Figure 4.10: User Profile Management

Architecture

User Profile Management (Fig. 4.10) is designed as a web service component that communicates with the profile and location repository through the JDBC API. It is assumed that only an administrator can access this service using the web form, whose responsibility is to administrate user profiles. User profile administration (Fig. 4.11) includes functions for creating, viewing, updating, and deleting the profile.

Profile repository (table 4.1) consists of the records described with the username, terminal, and profile column. It is assumed here that the user can apply a number of terminals and therefore the username and terminal are set as a primary key to identify the user profile. The profile is serialized in an RDF/XML form and stored as a blob in the repository.

Location repository (table 4.2) consists of the records containing the following fields: username, landmark's name, it's longitude and latitude, and a counter

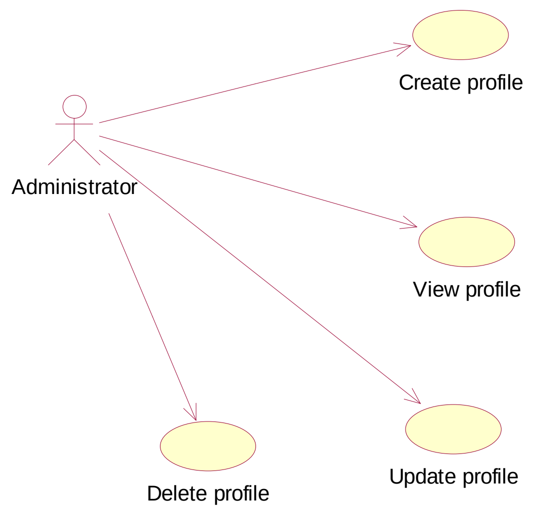


Figure 4.11: User Profile administration

Table 4.1: Profile repository

username	terminal	profile
A	Sony Ericsson P900	rdf/xml
A	PC	rdf/xml
B	Nokia 6600	rdf/xml

of user's visits to the landmark. The definition of user landmarks has followed the user's daily routine: he goes to *work* in the morning, after which he goes in the *shopping* or to *recreation*, and later on he returns *home*. Therefore the user landmark declaration has been limited to the finite set: {Home, Work, Shopping, Recreation}.

Table 4.2: Location repository

username	landmark	longitude	latitude	counter
A	Home	165248E	452931N	3
B	Work	162153E	451424N	2

Service interface

The User profile management web service provides the remote interface to the administrator and the location-based publish/subscribe web service to create, update and delete user profiles from the profile repository, as well as to store and retrieve user-defined landmarks from the location repository (Fig. 4.12).

```
 public interface ProfileMgmt extends Remote {

  String getProfile(
        String username, String terminal
  ) throws RemoteException;
  void storeProfile(
        String username, String terminal, String profile
  ) throws RemoteException;
  void deleteProfile(
        String username, String terminal
  ) throws RemoteException;
  Vector getComponent(
        String username, String terminal,
        String componentName) throws RemoteException;
  void updateComponent(
        String component, AttributeList attributes
  ) throws RemoteException;
  void defineLandmark(
        String username, String landmarkName,
        Coordinate coordinate) throws RemoteException;
  Landmark[] getLandmarks(
        String username
        ) throws RemoteException;
}
```

Figure 4.12: ProfileMgmt interface

4.3.4 User profile creation

In general, data about the user is stored in the so-called *user profile*. The information contained in the profile can be considered as contextual information in the sense that it describes the environment in which users desire to operate. As such, it represents only a small part of the information domain of context-aware systems.

Motivation for RDF

User can make requests for information that may not be relevant to him at a certain time, but will be when a particular event occurs, such as reaching a location of interest. If the location relevance of the information, together with the location's coordinates and the information validity period were structured in a metadata format, it would be possible to automatically retrieve the information relevant to the current context of the user. To this extent, location-aware information services could be seen as context-aware services, because a location change event could, for example, trigger an action of searching and disseminating the appropriate information set to the user. The triggering could be more personalized to user's needs if it would depend on several context parameters. The example is the user's availability for communication, as the user may not wish to be disturbed with the receiving of information, as he may set his availability for communication to "not available". These dependencies could be applied to all metadata elements, if they were represented in a well-structured, common data format, so the server could match them and produce the appropriate information set.

From the reasons discussed above and the context representation characteristics outlined in Section 3.2, RDF was chosen to create a data model comprising the properties that will be used in creation of user profiles.

User profile structure

User profiles are created using the profile ontology written in RDF [RDF03]. The ontology used to describe profile structure resembles the structure of the CC/PP schema [CCP03].

The user profile comprises the following components: *User Info, Terminal Capabilities, Presence, and Subscription* with the properties characterizing them (Fig. 4.13). Properties can contain static and dynamic values which can be supplied either automatically (by the services in the system) or can be input directly by the user.

The profile ontology reuses as much ontologies and specifications rather then reinventing them again, such as terminal capabilities properties from the WAP UAProf scheme [UaP99] and presence attributes form the Wireless Village Initiative [wvi02].

User personal information

The user personal data represent the static contextual information in the user profile structure. They describe the aspects of the system that usually do not change over time, for example: the username, user's e-mail address, mobile phone

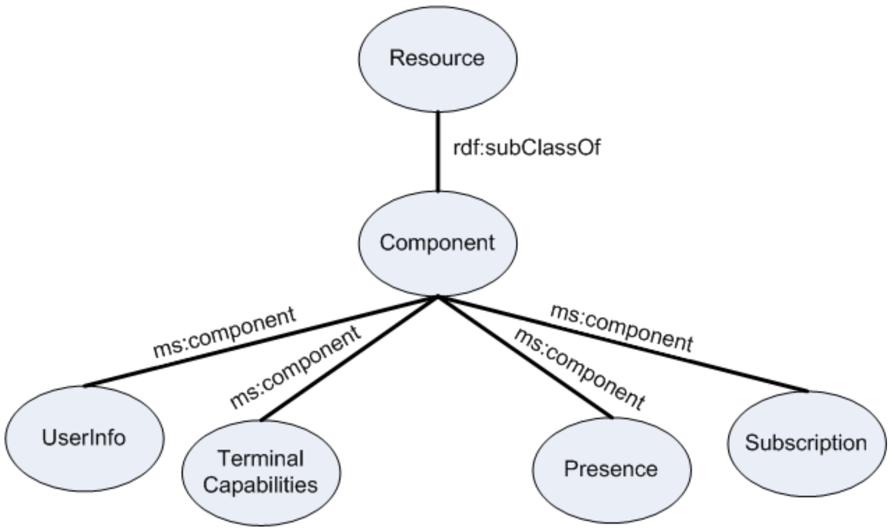


Figure 4.13: RDF graph

number, etc. The user can also prioritize the preferred contact means to emphasize the order in which he would like to receive information on the applied device (the delivery method). The profile ontology introduces a component called *UserInfo* consisting of the simple properties such as *username, password, email, msisdn*, as well as complex ones, such as *contacts*, pointing to the class *PrefferedContacts*, which in turn has properties for describing contact, such as *contactName* and *contactPriority.* These examples are serialized as follows:

```
<rdf:Description rdf:ID="UserInfo">
  <rdf:type rdf:resource="&rdfs;Class"/>
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdf:Description>

<rdf:Description rdf:about="username">
  <rdf:type rdf:resource="&rdf;Property"/>
  <rdfs:domain rdf:resource="#UserInfo"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Description>

<rdf:Description rdf:about="password">
  <rdf:type rdf:resource="&rdf;Property"/>
  <rdfs:domain rdf:resource="#UserInfo"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Description>

<rdf:Description rdf:about="e_mail">
  <rdf:type rdf:resource="&rdf;Property"/>
  <rdfs:domain rdf:resource="#UserInfo"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Description>
```

```
<rdf:Description rdf:about="msisdn">
  <rdf:type rdf:resource="&rdf;Property"/>
  <rdfs:domain rdf:resource="#UserInfo"/>
  <rdfs:range rdf:resource="&xsd;positiveInteger"/>
</rdf:Description>

<rdf:Description rdf:about="contacts">
    <rdf:type rdf:resource="&rdf;Property"/>
    <rdfs:domain rdf:resource="#UserInfo"/>
    <rdfs:range rdf:resource="#PrefferedContacts"/>
</rdf:Description>

<rdf:Description rdf:ID="PrefferedContacts">
    <rdf:type rdf:resource="&rdfs;Class"/>
    <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdf:Description>

<rdf:Description rdf:about="contactName">
    <rdf:type rdf:resource="&rdf;Property"/>
    <rdfs:domain rdf:resource="#PrefferedContacts"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Description>

<rdf:Description rdf:about="contactPriority">
    <rdf:type rdf:resource="&rdf;Property"/>
    <rdfs:domain rdf:resource="#PrefferedContacts"/>
    <rdfs:range rdf:resource="&xsd;positiveInteger"/>
</rdf:Description>
```

Terminal characteristics

The terminal capabilities data introduce the detailed information about user terminal hardware, software, and network characteristics when accessing services or some other content. The profile ontology defines component called *TerminalCapabilities* with the simple property *clientType* and reuses properties defined in WAP UAProf scheme [UaP99], such as: *Vendor* and *Model* for describing hardware characteristics. The client type property describes the type of the mobile terminal, which can be mobile phone, PDA, or PC.

```
<rdf:Description rdf:about=
"http://www.example.org/UserProfile#TerminalCapabilities">
  <ms:clientType>mobile_phone</ms:clientType>
  <uaprof:vendor>Nokia</uaprof:vendor>
  <uaprof:model>6600</uaprof:model>
  <ms:javaPlatform>
    <rdf:Bag>
      <rdf:li>CLDC/1.0</rdf:li>
      <rdf:li>MIDP/2.0</rdf:li>
    </rdf:Bag>
  </ms:javaPlatform>
</rdf:Description>
```

The declaration of namespaces used in the user profile document is the following:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:uaprof="http://www.wapforum.org/profiles/UAPROF/
                        ccppschema-20000405#"
         xmlns:ms="http://stella.zavod.tel.fer.hr:8080/RDF/myScheme.rdf#"/>
```

The terminal characteristics component can be further extended with other properties specified in WAP UAProf ontology to express information about displaying images, audio or video content, browser user agent, or wap capabilities. The intention of its use in the system was to provide minimum system information capable to discover user's applied terminal in order to provide the user with continuity of the service.

Location-based subscription

Information services require some user-defined preferences for the type of information that users want to receive. In the context of location-aware information services the location-based subscription is defined as a name of the topic used to publish and deliver content to interested parties, together with the location where the user would prefer to receive the published content. The landmark-based subscription is used to denote that the user can choose to receive his favorite content only when he is located on the selected landmark. A landmark consists of the name of the location and it's geographic coordinate (longitude, latitude). The following RDF serialization contains definition of the location-based subscription component, as well as its properties.

```
<rdf:Description rdf:about="Subscription">
  <rdf:type rdf:resource="&rdfs;Class"/>
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdf:Description>

<rdf:Description rdf:about="topic">
  <rdf:type rdf:resource="&rdf;Property"/>
  <rdfs:domain rdf:resource="#Subscription"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Description>

<rdf:Description rdf:about="landmark">
  <rdf:type rdf:resource="&rdf;Property"/>
  <rdfs:domain rdf:resource="#Subscription"/>
  <rdfs:domain rdf:resource="#Presence"/>
  <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Description>
```

Presence information

The presence information is introduced to describe user current state and activity in order to assist the application in providing him the relevant content and presenting it on the target terminal according to his current communication status. For example, if the user is currently in a meeting, and his presence status is "busy", the content delivery service will not send him the time-sensitive information. The presence attributes are reused from the Wireless Village Initiative [wvi02] and contain the following information: *the user terminal status (busy/idle/detached)*, his current *landmark (Home/Work/Shopping/Recreation)*, and information about his *availability for communication (available/not available/discreet)*. The presence component use is demonstrated on the following example:

```
<rdf:Description rdf:about=
"http://www.example.org/UserProfile#Presence">
  <ms:landmark>Work</ms:landmark>
  <ms:onlineStatus>busy</ms:onlineStatus>
  <ms:userAvailability>available</ms:userAvailability>
</rdf:Description>
```

Presence properties are dynamic, meaning that their values change in time. They are collected from different parts of the system, such as location and status server, as well as user equipment, and their change triggers the update of the presence information in the user profile.

Chapter 5

System implementation

This chapter presents the system implementation of the proposed architecture described in the previous chapter. The system consists of a MIDlet client and a server represented by two Java web-based applications: the location-aware content delivery service and the user profile management service, both of them composed of appropriate components and built in a Java servlet and Java Server Pages (JSP) technology (Fig. 5.1).

The client application enables the user to declare his landmarks, publish and subscribe to (location-aware) content, and specify his preferences regarding receiving the desired content on his mobile terminal. The server side of the system was briefly described in the previous chapter, and it won't be discussed again.

The chapter is structured as follows: Section 5.1 gives an overview of the system implementation. The focus of the chapter is put on the implementation of the client application, to show the user's perspective and the interaction with the system, which is described in Section 5.2. Section 5.3 outlines the implementation of the user profile management service from an administrator's point of view. The system deployment is described in Section 5.4.

5.1 Implementation overview

The communication between client and server is performed through XML over HTTP. Client requests are received by the HTTP request handler, which depending on the required action, routes them to one of the following component handlers: positioning handler, map handler, content provider handler, location handler, status handler, and profile handler. The appropriate component handler, on the other hand, communicates with one of the following components: location server, map

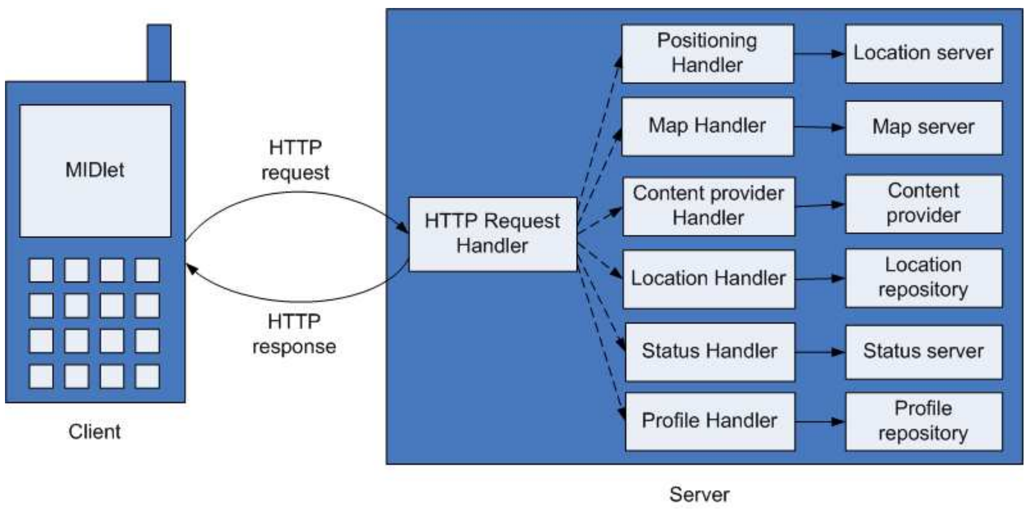


Figure 5.1: System implementation

server, content provider, location repository, status server, and profile repository. The selected component handler transforms the request into the form that is more appropriate for the target component, and waits for the result from the component. When available, it forwards the received result to the user.

5.2 Client application

The client application runs in a MIDP-enabled phone. It provides a user interface for logging into the system, landmark declaration, creating and publishing the location-aware content, as well as subscribing to the desired content.

The MIDlet class diagram is presented on Fig. 5.2. The BeginMIDlet first creates a HttpPoster, passing it the URL of the server. The BeginMIDlet then creates a MenuScreen as a starting point for the user to choose a desired functionality from the menu, and a LoginScreen, passing it a reference back to the HttpPoster, the MenuScreen, and the reference to itself for callbacks.

The LoginScreen is displayed first (Fig. 5.3). It calls the HttpPoster to send a login request to the server to authorize the user for the application use; it passes the HttpPoster the reference to itself as a HttpPosterListener, so the HttpPoster can use callbacks when the response arrives. Since the HttpPoster does not see a LoginScreen but the HttpPosterListener, it does not depend on the LoginScreen, and therefore the HttpPoster-HttpPosterListener combination is used by other screen classes, too.

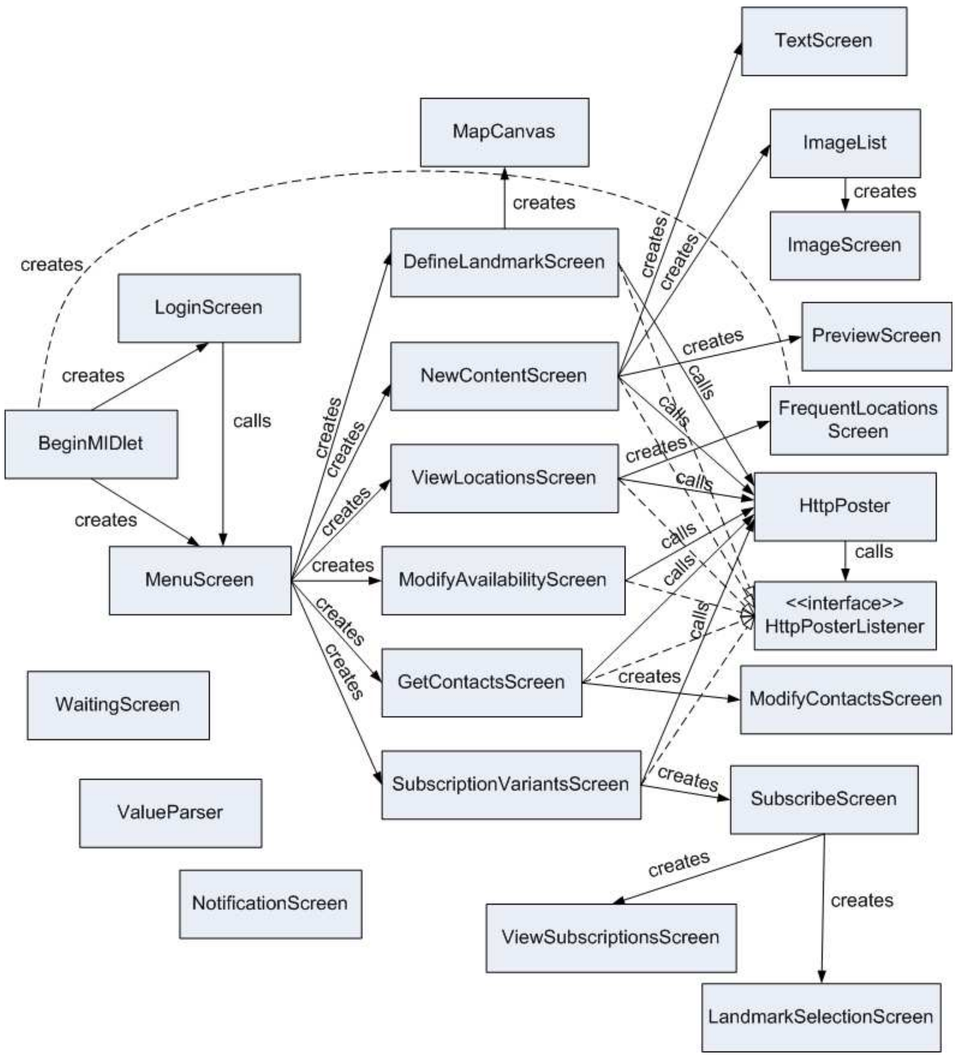
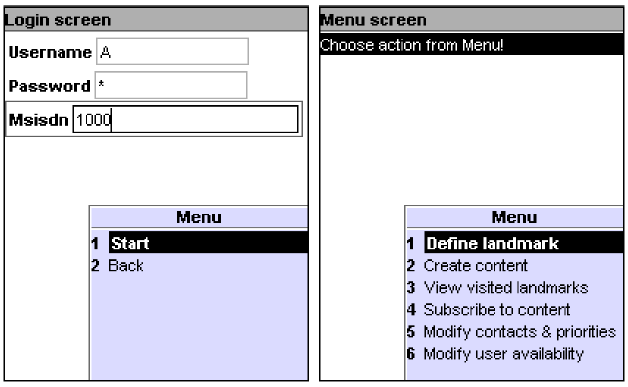


Figure 5.2: MIDlet class diagram

The LoginScreen uses a WaitingScreen as an intermediate screen when waiting for the login response to arrive. NotificationScreen is used to display notifications to the user. The LoginScreen uses a ValueParser to parse the application response, that it receives from the server. All screens that communicate with the server do this; this detail is not repeated in the descriptions below.

When the login completes successfully, the LoginScreen displays the MenuScreen (Fig. 5.3). When the user chooses an action from menu, the MenuScreen creates one of the following screens: a DefineLandmarkScreen, a NewContentScreen, a ViewLocationsScreen, a SubscriptionVariantsScreen, a ModifyAvailabilityScreen, or a GetContactsScreen, dependently of the chosen action (passing it a reference to the BeginMIDlet, to HttpPoster and itself for callbacks).


Login screen

Username A

Password *

Msisdn 1000

Menu

1 Start

2 Back

Menu screen

Choose action from Menu!

Menu

1 Define landmark

2 Create content

3 View visited landmarks

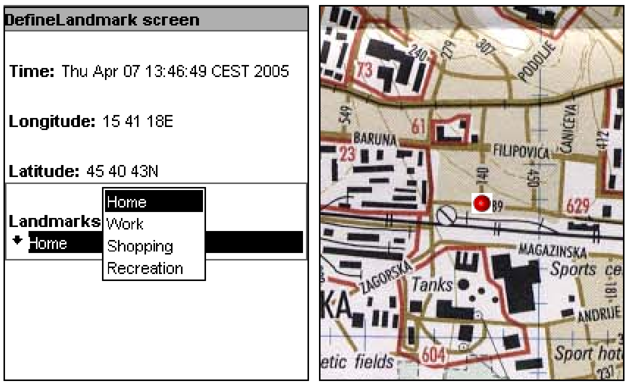
4 Subscribe to content

5 Modify contacts & priorities

6 Modify user availability

Figure 5.3: Login and menu screen

At the beginning of the application, it is assumed that the user first defines his landmarks. The positioning request is sent to the location server and when the response arrives, the DefineLandmarkScreen is created and displayed (Fig. 5.4). It presents the user geographical coordinate pair (longitude, latitude) with the time when the position is determined, and offers him a choice group of landmarks to select from. The screen contains commands for viewing the map of the located area and submitting the landmark to the system for storing it in the location repository.


DefineLandmark screen

Time: Thu Apr 07 13:46:49 CEST 2005

Longitude: 15 41 18E

Latitude: 45 40 43N

Landmarks Home

Home

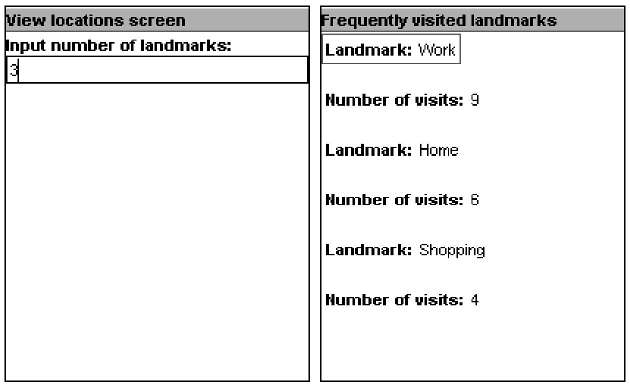
Work

Shopping

Recreation

Figure 5.4: Declare landmark

The DefineLandmarkScreen creates a MapCanvas and sends the request for map to the server, passing it the screen size of the mobile terminal. The system will, after receiving this request, dynamically crop the map of the city according to the located user position and adapt its size to mobile terminal's screen dimensions. The application response will contain the url of the desired map location, that will be used to retrieve the map image from the web server, which will be displayed on the MapCanvas (Fig. 5.4). After viewing his position on the map, user can decide whether he wants to assign this location to one of the provided landmarks. If true, the DefineLandmarkScreen will send the request to the server to submit the selected landmark to the system.



View locations screen

Input number of landmarks: 3

Frequently visited landmarks

Landmark: Work

Number of visits: 9

Landmark: Home

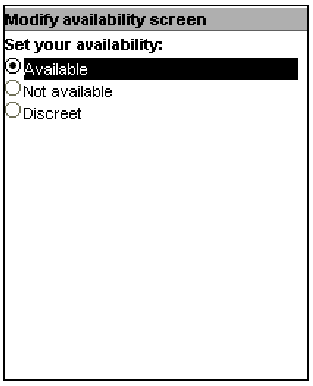
Number of visits: 6

Landmark: Shopping

Number of visits: 4

Figure 5.5: View visited landmarks

Due to the fact that the positioning system periodically sends a location push to the system and locations that correspond to user landmarks' coordinates are stored in the location repository, the user has an option to preview his most frequently visited landmarks. In that case, the ViewLocationsScreen is created and displayed to the user (Fig. 5.5). It offers the user a text field to input the number of the most visited landmarks he wants to check, which will be sorted upon the received request according to the counter of user's visits to the specified location. This number is then sent within the request to the server, and the answer reception will trigger the ViewLocationsScreen to create a FrequentLocationsScreen that will list the sorted landmarks with their corresponding counters of visits (Fig. 5.5).



Modify availability screen

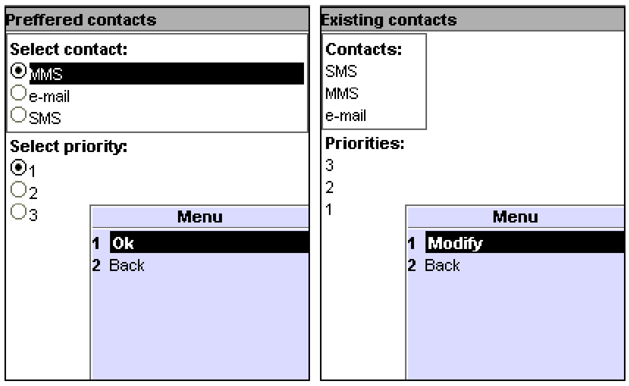
Set your availability:

[✓ Available] Not available Discreet

Figure 5.6: Modify availability

If the user wants to modify the setting for his availability for communication, he can select this action from menu, which will initiate the MenuScreen to create the ModifyAvailabilityScreen (Fig. 5.6). The ModifyAvailabilityScreen offers availability values to the user from which he can choose and modify his current setting. It sends then the new value in the request to the server to update the user profile in the profile repository.

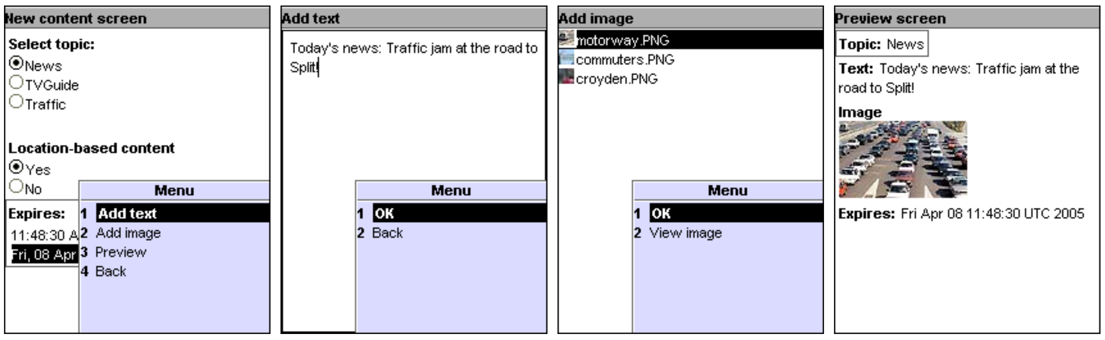
The user has an option to modify his contacts and priorities, meaning that he can specify the means of content delivery and prioritize them. After selecting this action from menu, the MenuScreen creates the GetContactsScreen (Fig. 5.7), that retrieves the list of existing contacts and their priorities from the user profile and offers command for modifying this setting. Upon selecting this command, the GetContactsScreen will create a ModifyContactsScreen that offers possible contacts and priorities to be assigned to them. When selected, values are inserted into the request and submitted to the server.


Preffered contacts
Select contact: [✓ MMS] e-mail SMS
Select priority: [✓ 1] 2 3
Menu
1 Ok
2 Back

Existing contacts
Contacts:
SMS
MMS
e-mail
Priorities:
3
2
1
Menu
1 Modify
2 Back

Figure 5.7: Get contacts

If the user chooses to create and publish the new content, the MenuScreen will create a NewContentScreen (Fig. 5.8), which offers the user a choice group of topics, as well as an option to determine if the content will be location-based or not, and an expires field that denotes the content validation time, to specify how the content will be published. The screen offers commands for adding text, image, and preview the content, before it is submitted for publishing. Therefore, a TextScreen, an ImageList and/or a PreviewScreen are created when needed (Fig. 5.8). The TextScreen provides a text box for entering the textual part of the content. The ImageList offers a set of images that can be included in the content. It creates an ImageScreen when the user wants to have a look at the image, before making a selection. The PreviewScreen makes a preview of the user's created


New content screen
Select topic:
● News
○ TVGuide
○ Traffic
Location-based content
● Yes
○ No
Expires:
11:48:30 A
Fri, 08 Apr
Menu
1 Add text
2 Add image
3 Preview
4 Back

Add text
Today's news: Traffic jam at the road to Split!
Menu
1 OK
2 Back

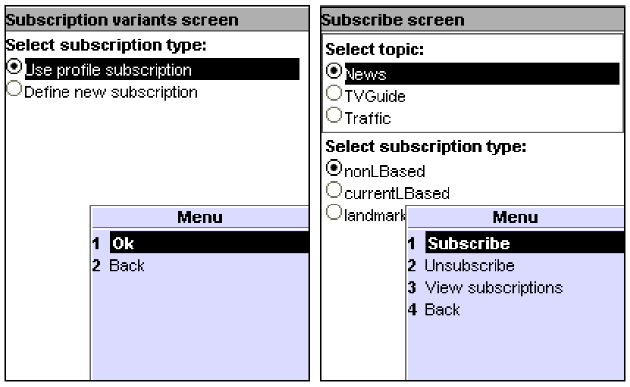
Add image
motorway.PNG
commuters.PNG
croyden.PNG
Menu
1 OK
2 View image

Preview screen
Topic: News
Text: Today's news: Traffic jam at the road to Split!
Image
Expires: Fri Apr 08 11:48:30 UTC 2005

Figure 5.8: Create content

content and after pressing the command for publishing the content, encapsulates it in the request and sends the request to the server.

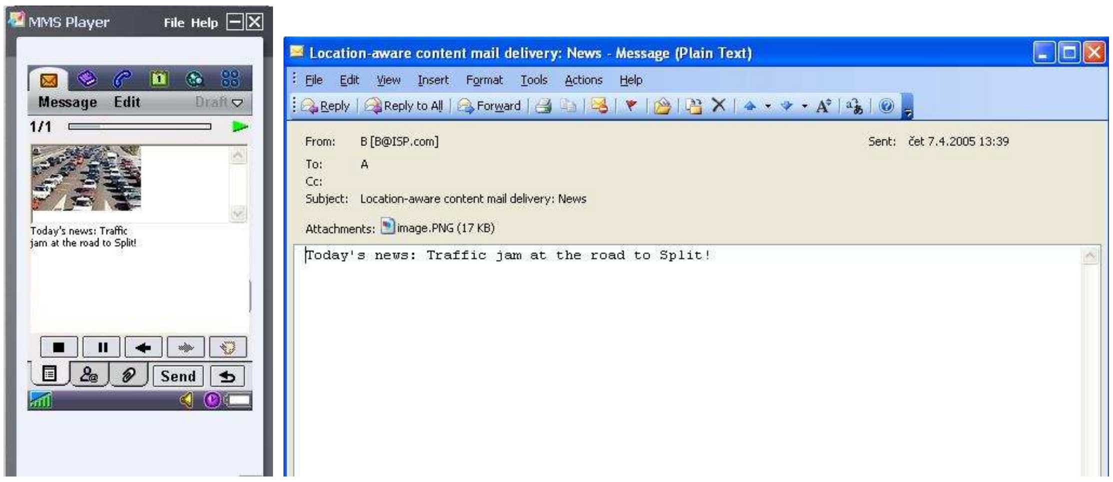
When the user chooses an option from menu to subscribe to the content, the SubscriptionVariantsScreen will be created and displayed (Fig. 5.9). It enables the user to select the subscription variant which determines the way he wants to subscribe to the desired content, i.e. he can subscribe either using the subscription defined in the user profile or he can choose to define a new subscription.


Subscription variants screen
Select subscription type:
● Use profile subscription
○ Define new subscription
Menu
1 Ok
2 Back

Subscribe screen
Select topic:
● News
○ TVGuide
○ Traffic
Select subscription type:
● nonLBased
○ currentLBased
○ landmark
Menu
1 Subscribe
2 Unsubscribe
3 View subscriptions
4 Back

Figure 5.9: Subscribe screen

After the completed selection, the SubscriptionVariantsScreen creates either a NotificationScreen, if the subscribing procedure using the subscription defined in the user profile has been successfully completed, or a SubscribeScreen (Fig. 5.9) with three possible subscription types: the current location-based, the landmark-based, and the non location-based subscription, and following topics to choose from: News, Traffic and TV Guide. If the landmark-based subscription type is selected, the Subscribe screen creates a LandmarkSelectionScreen which offers the user to choose the landmark where he would prefer to receive the desired content.


MMS Player
File Help
Message
Edit
Draft
1/1
Send
Location-aware content mail delivery: News - Message (Plain Text)
File Edit View Insert Format Tools Actions Help
Reply
Reply to All
Forward

Today's news: Traffic jam at the road to Split!

From: B [B@ISP.com] Sent: čet 7.4.2005 13:39
To: A
Cc:
Subject: Location-aware content mail delivery: News
Attachments: image.PNG (17 KB)

```
Today's news: Traffic jam at the road to Split!
```

Figure 5.10: Content delivered via MMS and e-mail

After providing the needed data, the subscribe request is sent to server. From that moment, the content is expected to arrive on the applied terminal in the form specified in the user profile (as SMS, MMS, or e-mail) (Fig. 5.10).

The SubscribeScreen also enables the user to check his existing subscriptions, before he defines a new one, or to unsubscribe from one of the existing subscriptions. In the former case, a ViewSubscriptionsScreen is created and the list of valid user subscriptions (which are stored in the user profile) is presented on display. In the latter case, the request for unsubscribe from the specified subscription is submitted in request to the server.

5.3 User Profile Management Service Implementation

The user profile management service implements a Web interface defined in Section 4.3.3. using Java RMI. The service is accessed through a web interface that is created for the administrator to access user profiles stored in the profile repository. At the beginning, a windows authentication (Fig. 5.11) using username and password is required. Due to the personal nature and privacy of data stored in the user profile, only the administrator has a privilege and rights to preview and modify them.

After the successful login, the administrator gets a menu page where from he can choose one of the following actions: to create a new user profile, or to preview,

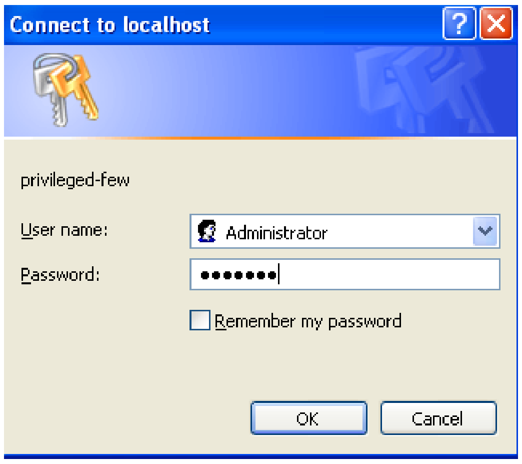


Figure 5.11: Authentication

update, and delete the existing profile from the repository (Fig. 5.12).

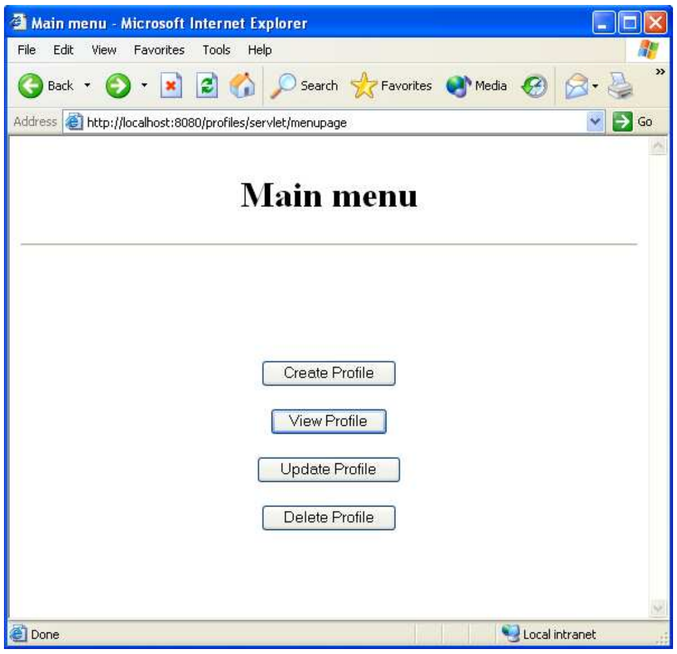

Main menu - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/profiles/servlet/menupage Go
Main menu
Create Profile
View Profile
Update Profile
Delete Profile
Done
Local intranet

Figure 5.12: Main menu

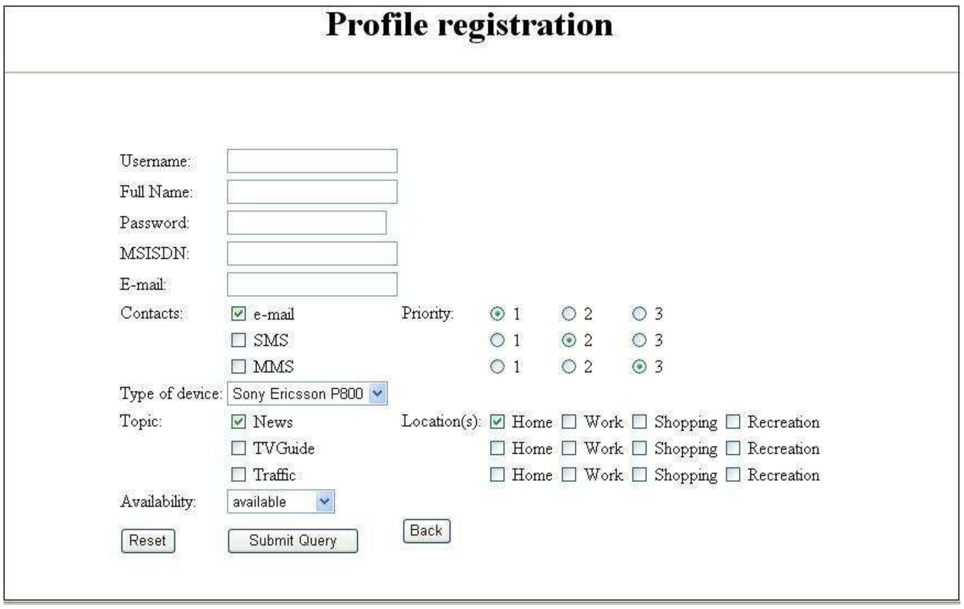
Creating a new profile opens a new page with a web form to be filled with a user's data which is required for user profile creation (Fig. 5.13). The form contains:

- the user's personal information (username, full name, password, msisdn)
- the content delivery information (the preferred delivery method with the

assigned priority, the type of device, the content he is interested in receiving, as well as landmarks where he would like to receive it on his mobile phone)

- the presence information (his availability for communication)

After retrieving the required data, the service creates a user profile document, assigns these values to properties obtained from namespaces specified in the document, and attaches them to the user profile.



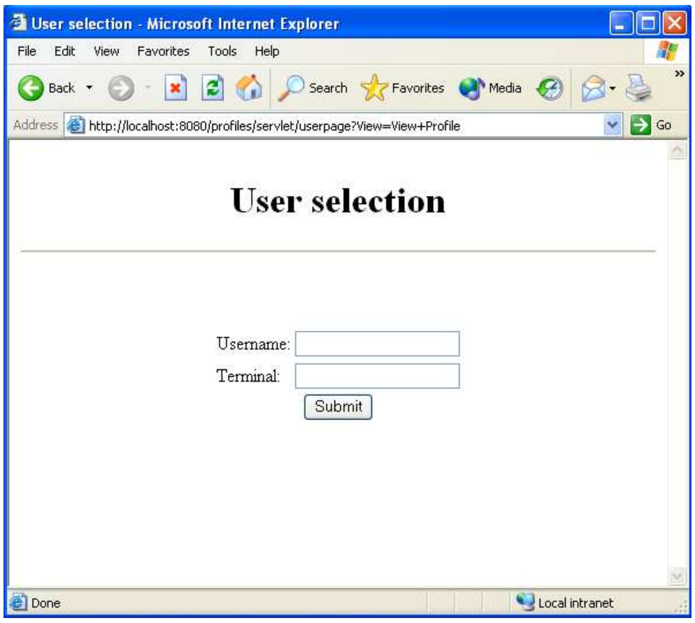
Profile registration

Username:						
Full Name:						
Password:						
MSISDN:						
E-mail:						
Contacts:	✓ e-mail	Priority:	● 1	○ 2	○ 3	
	☐ SMS		○ 1	● 2	○ 3	
	☐ MMS		○ 1	○ 2	● 3	
Type of device:	Sony Ericsson P800					
Topic:	✓ News	Location(s):	✓ Home	☐ Work	☐ Shopping	☐ Recreation
	☐ TVGuide		☐ Home	☐ Work	☐ Shopping	☐ Recreation
	☐ Traffic		☐ Home	☐ Work	☐ Shopping	☐ Recreation
Availability:	available					

Figure 5.13: Creating a new profile

The profile is generated using Jena, a Java API for RDF. Jena represents an open-source Java framework for building Semantic Web applications [Jen]. It provides a programmatic environment for RDF, RDFS, and OWL, including a rule-based inference engine. It is grown out of work with the HP Labs Semantic Web Programme [HP]. The Jena Framework includes in its current version (2.0) the following: an RDF API, an OWL API, in-memory and persistent storage, an RDQL - a query language for RDF, and integrated parsers and writers for an RDF/XML, a N3, and N-triples. The RDF/XML, the N3, and N-triples are different serialization techniques for reading and writing an RDF model [RDF]. The querying functionality is used when searching for or updating properties in the user profile. Jena facilitates understanding of the RDF data model to a developer as opposed to the learning from theory.

When generated, the user profile is stored in the profile repository together with the user's username and his terminal identifier. The user profile's structure was briefly described in Section 4.3.4.


User selection - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address http://localhost:8080/profiles/servlet/userpage?View=View+Profile
Go
Done
Local intranet

User selection

Username:

Terminal:

Submit

Figure 5.14: User selection

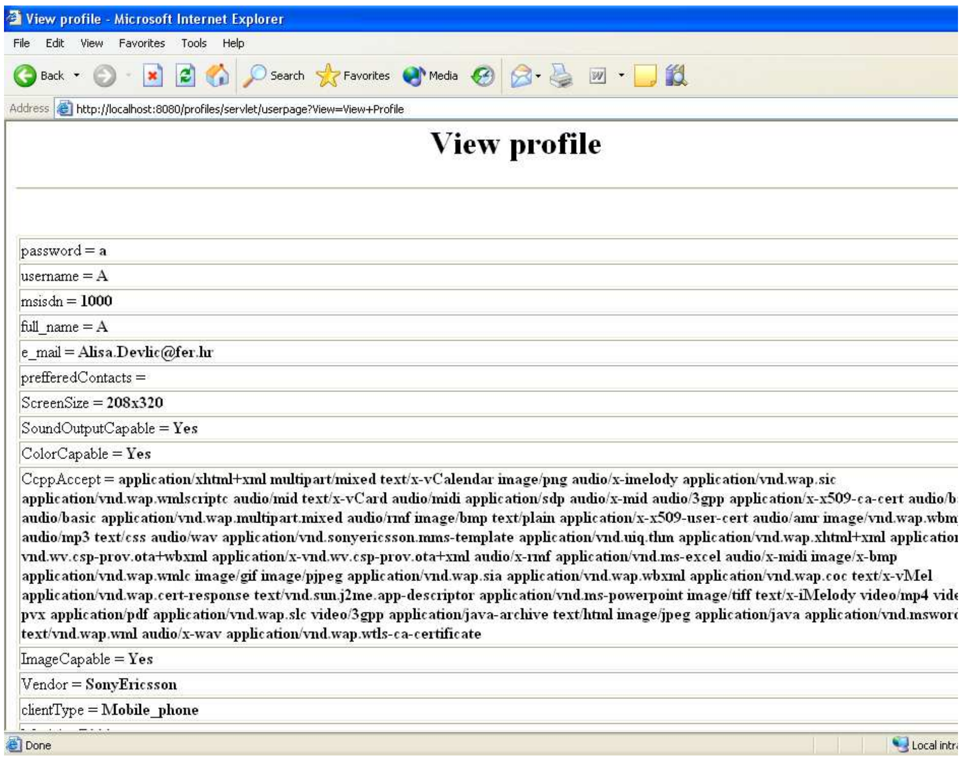
If the administrator chooses to preview the existing profile, he has to enter the username and the type of device (terminal identifier) that will identify the profile that needs to be retrieved from the database (Fig. 5.14).

Properties and assigned data values from the profile are then displayed in the table in the view profile page (Fig. 5.15).

The action of updating an existing profile is very similar to one of creating a new profile, with a difference that the username and the terminal identifier of the user, whose profile is to be modified, need to be passed to the service by filling in the user selection page (Fig. 5.14).

The profile generation form is then presented, but filled with values from the existing profile. The fields in the form are editable, and every modification entered by the administrator will be stored in the profile.

The deleting operation also requires the username and the terminal identifier entered in the user selection page, followed by a question to the administrator to confirm that the profile can be deleted from the database.



View profile - Microsoft Internet Explorer

http://localhost:8080/profiles/servlet/userpage?View=View+Profile

View profile

password = **a**

username = **A**

msisdn = **1000**

full_name = **A**

e_mail = **Alisa.Devlic@fer.hr**

prefferedContacts =

ScreenSize = **208x320**

SoundOutputCapable = **Yes**

ColorCapable = **Yes**

CcppAccept = **application/xhtml+xml multipart/mixed text/x-vCalendar image/png audio/x-imelody application/vnd.wap.sic application/vnd.wap.wmlscriptc audio/mid text/x-vCard audio/midi application/sdp audio/x-mid audio/3gpp application/x-x509-ca-cert audio/b audio/basic application/vnd.wap.multipart.mixed audio/rmf image/bmp text/plain application/x-x509-user-cert audio/amr image/vnd.wap.wbm audio/mp3 text/css audio/wav application/vnd.sonyericsson.mms-template application/vnd.uiq.thm application/vnd.wap.xhtml+xml applicatio vnd.wv.csp-prov.ota+wbxml application/x-vnd.wv.csp-prov.ota+xml audio/x-rmf application/vnd.ms-excel audio/x-midi image/x-bmp application/vnd.wap.wmlc image/gif image/pjpeg application/vnd.wap.sia application/vnd.wap.wbxml application/vnd.wap.coc text/x-vMel application/vnd.wap.cert-response text/vnd.sun.j2me.app-descriptor application/vnd.ms-powerpoint image/tiff text/x-iMelody video/mp4 vide pvx application/pdf application/vnd.wap.slc video/3gpp application/java-archive text/html image/jpeg application/java application/vnd.mswor text/vnd.wap.wml audio/x-wav application/vnd.wap.wtls-ca-certificate**

ImageCapable = **Yes**

Vendor = **SonyEricsson**

clientType = **Mobile_phone**

Done

Figure 5.15: View an existing profile

5.4 System deployment

Having in mind the service architecture for 3G mobile networks discussed in Section 4.3.1, system components are deployed across several nodes in the network (Fig. 5.16).

Components are the following:

- the city map provider - hosts the city map of Zagreb on the Apache web server;
- the content provider - provides a publish/subscribe middleware using the Joram messaging system;
- the service provider - deploys the location-aware content delivery service on the Tomcat servlet container;
- the mail server - uses pop.tel.fer.hr as an incoming mail server;
- the user equipment (Sony Ericsson P900 smart phone) - has installed a client application written in J2ME for CLDC1.0/MIDP2.0 devices;

- the mobile network operator - deploys the user profile management service on the Tomcat servlet container that communicates with the location and profile repository (built on MySQL server), as well as hosts a MMS-C, a SMS-C, and a Mobile positioning system, to provide a multimedia messaging service, a short messaging service and a positioning functionality to mobile users.

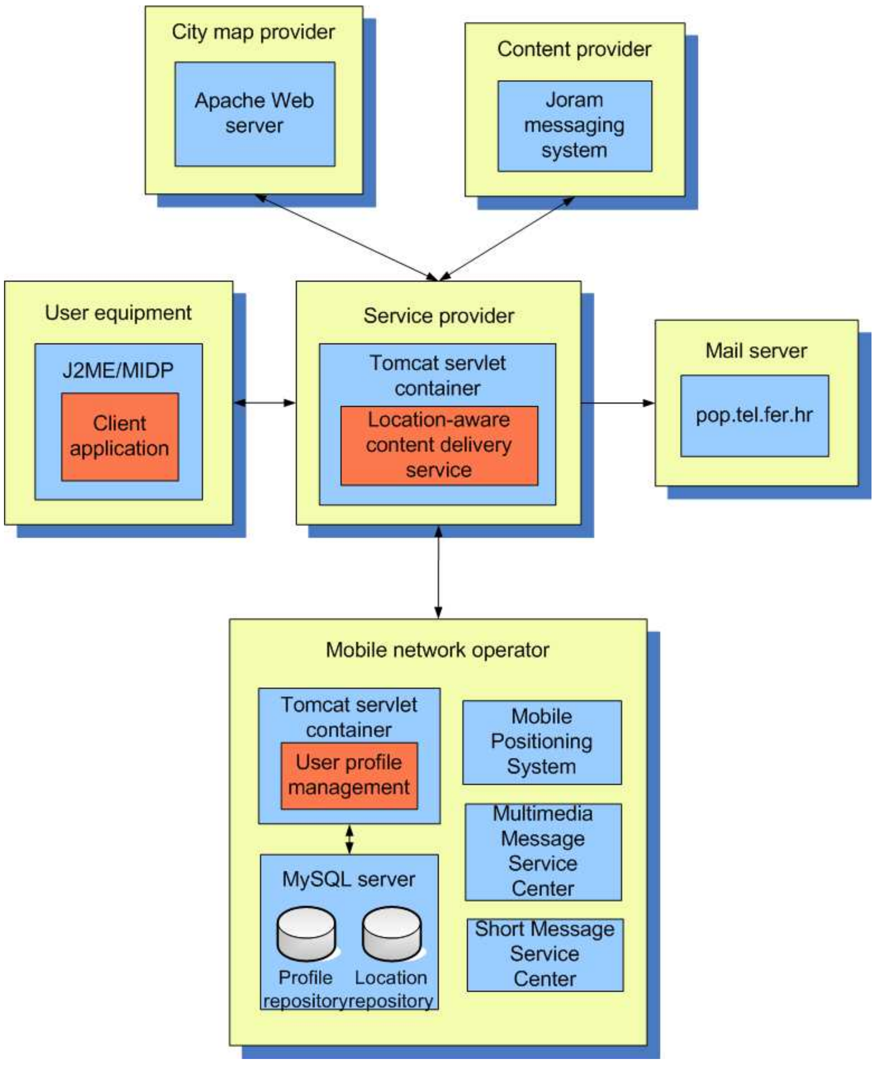


Figure 5.16: Deployment

Chapter 6

Service provisioning

In the telecommunications domain, a *provisioning* is defined as the setting in place and the configuring of hardware and software required to activate a telecommunications service for a customer. In many cases hardware and software may already be in place, and the provisioning entails only configuration tasks such as creating (or modifying) a customer record in a database and associating it with the service(s) and the service level for which the customer has been subscribed.

Another definition specifies the provisioning as an act of acquiring the telecommunication service from the submission of the requirement through the activation of the service, including everything necessary to set up the service, such as an equipment, a wiring, and a transmission.

In a wireless environment, the provisioning refers to the service activation and involves programming various network databases with the customer's information.

An agent paradigm is discussed and applied in this work for the service provisioning. Agents are general-purpose *carriers* of programs, rather than individual and application-specific programs. Agent-based paradigm enables an easy development of autonomous software agents that can discover, retrieve, install, execute, and monitor services dynamically, without shutdown/restart, as well as retool clients by loading new programs on the fly. This is very important feature, used for delivering of services on the air to clients in the mobile network. As it was mentioned earlier, software agents can play a brokerage role to match user requirements against service capabilities, and provide the user with the service that best suites his needs.

Furthermore, distributed software in the network can be managed by software agents. Remote operation management includes operations, such as software installation, upgrading, and testing on the remote target system. These operations are performed by mobile agents. Mobile agent is a program that represents its user in the network and can autonomously migrate from node to node to perform some actions on behalf of its user. Mobile agents with an associated task are injected in the network that allows them to roam toward the target node, and return to their home node to report results.

The chapter is organized as follows: Section 6.1 gives an overview of the service provisioning in the mobile network. An Over-The-Air provisioning, the current recommended practice to support the service deployment on mobile devices, is explained in the Section 6.1.1. Section 6.2 outlines the proposed provisioning of the location-aware content delivery service, physically separated in two parts. The *client application* on the mobile terminal will be provisioned using the semantic agent and its matchmaking capabilities, and it is described in Section 6.2.1. The *server part* of the service will be migrated, installed, and run on the remote network node using the Remote Maintenance Shell (RMS), a framework developed for the remote software maintenance and execution of above specified remote operations, outlined in Section 6.2.2.

6.1 Service Provisioning in the Mobile Network

As the functionality of mobile devices grows at an increasing rate, configuring and maintaining mobile applications and services becomes a complex and time-consuming task. For instance, enabling WAP, GPRS, MMS, and data connectivity requires configuration of multiple settings. Even with limited features of today, many customers find it difficult to configure their mobile devices. Operators should ensure that the phone configuration and the service delivery and deployment on the customer's mobile device can be done quickly and easily.

An effective and widespread adopted, standard-accepted mechanism for service provisioning and device management in the mobile network is an Over-The-Air (OTA) provisioning. The OTA mechanism is a process of a remote management of device settings and applications, that no longer requires from users to change their devices and go to a physical location of the service provider to subscribe to services of interest. For mobile network operators and service providers this reduces the cost and the complexity of provisioning services and provides a fast and easy way to introduce new services, as well as manage provisioned services by dynamically adjusting to network changes.

The following text gives a brief overview of an OTA provisioning.

6.1.1 Over-The-Air Service Provisioning

The Over-The-Air (OTA) Service Provisioning is a process of provisioning mobile station (user equipment) operational parameters over the air interface. From the mobile client's perspective, OTA represents an ability to find an interesting application on the Web and to initiate its download over a mobile network. It is intended for deploying Mobile Information Device Profile (MIDP) applications (referred to as MIDlet suites) on J2ME/MIDP-enabled devices, but can be used to provision other types of content, too.

Both MIDP 1.0 and MIDP 2.0 devices support OTA provisioning. The first version of MIDP OTA specification appeared after the MIDP 1.0 specification, as a recommended practice [ota01], not as part of the specification itself. Many devices currently support the MIDP 1.0 recommendation.

MIDP 2.0 improved the MIDP 1.0 recommendation and made it part of the specification [ota02]. Because it is now part of the standard, all future MIDP devices will support OTA in a consistent, standard way. It's a great benefit to be able to distribute applications to many different mobile devices all in the same way.

In its simplest form, OTA is illustrated in the Figure 6.1 [Ort02].

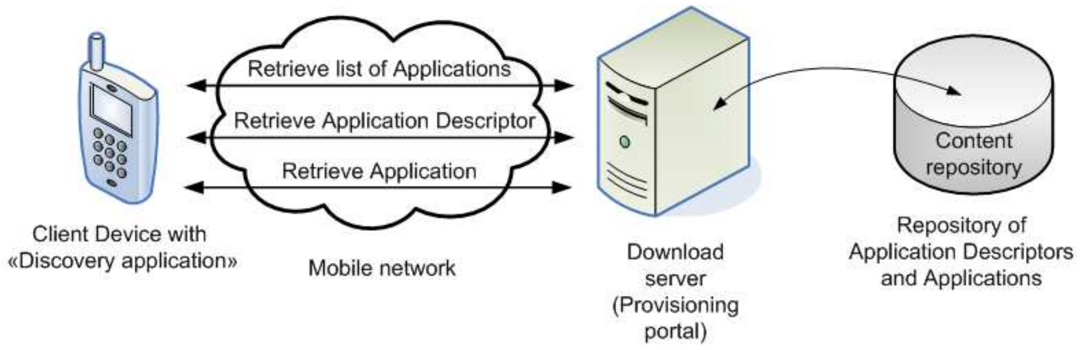

Retrieve list of Applications
Retrieve Application Descriptor
Retrieve Application
Content repository
Client Device with «Discovery application»
Mobile network
Download server (Provisioning portal)
Repository of Application Descriptors and Applications

Figure 6.1: A simplified view of OTA provisioning

The following entities participate in the process:

- **Client device with a discovery application**: devices must have software that allows them to locate MIDlet suites at a particular provisioning portal on the network, and to choose which applications to download - this software is referred to as a *discovery application (DA)*. In some cases, discovery will be performed using a device's resident browser (e.g. WAP), which is the same for MIDP OTA, as long as a common provisioning protocol with the download server is HTTP;

- **Mobile network**: this can be any mobile network that includes a radio network and a WAP gateway. It has two main functions: it provides properly formatted menus, often written in WML or HTML, that list applications currently available for download, and it provides access to applications;
- **Download server**: it is also called a provisioning portal, and represents a host, visible in the network, that typically runs a web server and has access to a content repository;
- **Content repository**: as the name implies, it is the repository of application descriptors and applications that are available for download.

In the actual use, an OTA provisioning system is not so simple. It encompasses a content publication and management, an access control, an installation (and upgrading) of applications, and a tracking the use of applications (content) for billing purposes. Figure 6.2 illustrates the detailed view of an OTA provisioning system:

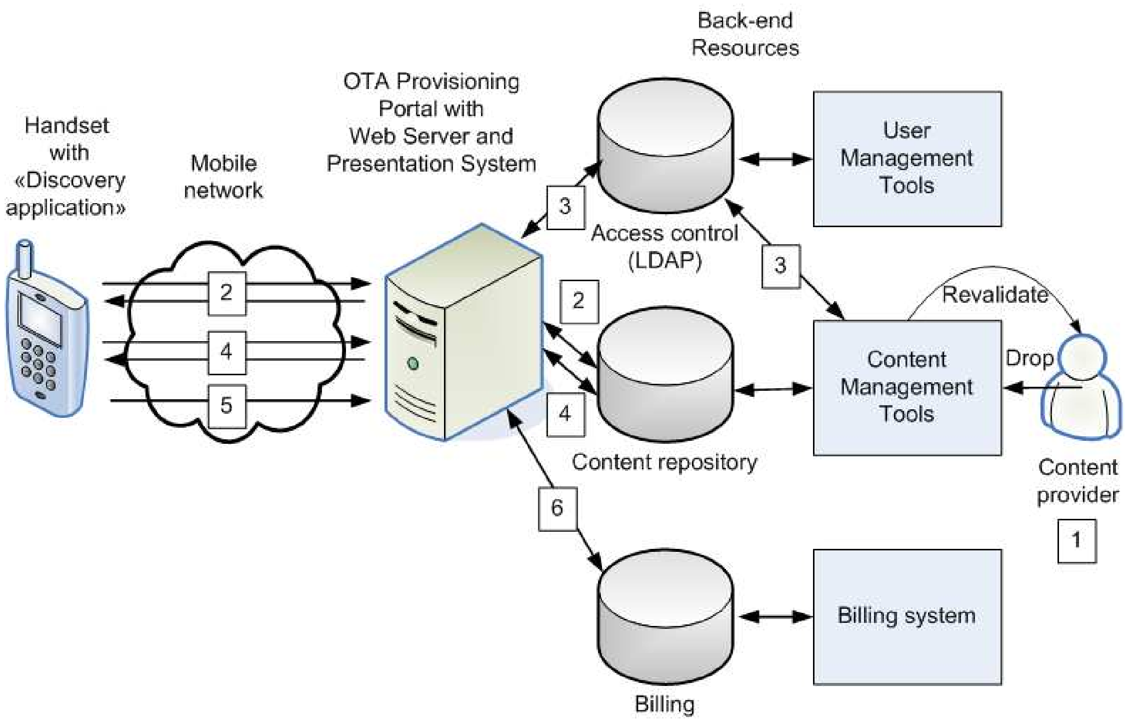


Figure 6.2: A detailed view of OTA provisioning

The detailed provisioning process follows the following steps:

- **1. Content management**: The server-side software manages the repository, typically a database, and supports content versioning, ways for third-party developers to drop their applications, and so on. Sometimes applications need to be certified before being available for OTA;
- **2. Content discovery**: The user points the discovered application to a download portal, that accesses the application repository and provides the properly formatted menu of the available content and applications;
- **3. Authenticate**: If the provisioning server supports an authentication module, the user is authenticated before gaining access to repository;
- **4. Application retrieval and installation**: Once the authentication is completed, downloading the application is performed in two parts, handled by the application management system (AMS). AMS is the software on the device that manages the download, installation, execution, and removal of application and other resources in device. If an application description exists (in the form of JAD file), AMS downloads it from the provisioning server's repository. Based on information found in the downloaded application descriptor, AMS automatically downloads the application (the MIDlet suite JAR) from the repository. If the application is retrieved successfully, the installation is automatic;
- **5. Confirmation**: AMS sends a confirmation status to indicate whether the installation succeeded or failed;
- **6. Tracking**: The confirmation status can be used to track the use of application, for example for billing purposes. A billing system is often integrated into the provisioning server.

During the download and the installation process, the user has an option to control the download, determine which versions of software are installed, remove MIDlet suites once downloaded, and obtain information about the MIDlet suite(s) on the device.

The OTA lifecycle has been summarized in Figure 6.3. It starts with the user instructing the device (using the DA) to search for an application of interest on the provisional portal in the network. On finding one, the user selects the application to download and install. After the installation, the application can be executed,

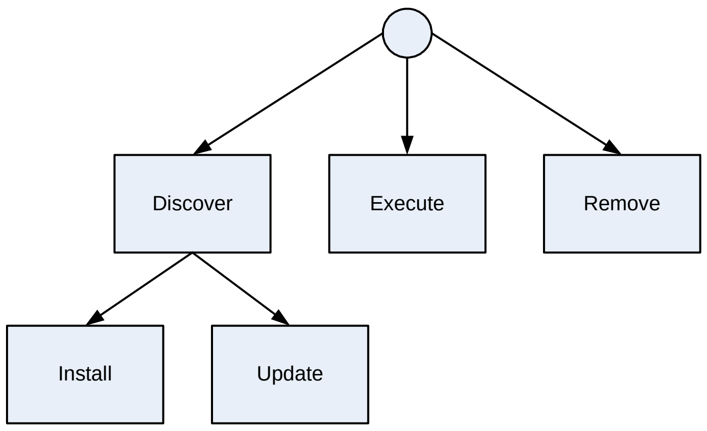


Figure 6.3: OTA application provisioning lifecycle

updated, or removed from the device. All stages of the OTA lifecycle are managed by AMS.

6.2 Location-aware service provisioning

An integration of public telecommunication and internet information services has created an environment for advanced personalized mobile services that can be executed in heterogeneous systems. The personalization concept refers to mechanisms of discovering, selecting, and combining services according to user preferences. The requirements for user mobility in heterogeneous environments include the delivery of the same set of services independent of the user current location and his applied terminal, thus requiring the adaptation of content according to user requirements and terminal capabilities.

Also, in the network with a number of remote nodes with a different set of installed services, there is a problem of service provisioning, as well as remote software maintenance and monitoring. In the context of remote software maintenance, service provisioning refers to the delivery of the service on the remote node, the monitoring of its execution, and the management of the service in the runtime. The service delivery encompasses remote operations, such as service migration and installation, while monitoring and service management involve operations of starting/stopping the service execution, the service removal, and its upgrade.

These concepts are applied on the location-aware content delivery service, that delivers location-aware content chosen by the user to his applied terminal. The service consists of the client and server part, and its architecture and implementation are explained in Sections 4 and 5, respectively.

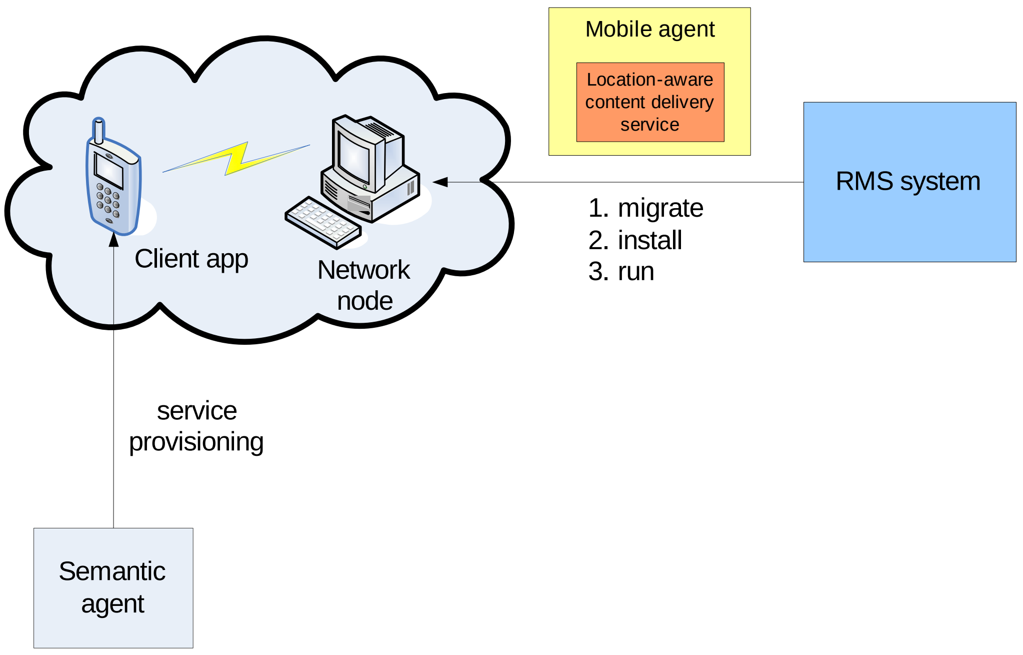


Figure 6.4: Location-aware service provisioning

The service provisioning is performed in two different parts (Fig. 6.4). First part refers to the utilization of the semantic agent for provisioning of the client application on the mobile device. Second part concerns the delivery of the server part of location-aware content delivery service on the remote node. Software delivery operations include migration, installation, and service execution on the remote node, using mobile agents. A Remote Maintenance Shell (RMS) is a framework developed as a multi-agent system supporting remote software operations, such as software installation, starting, stopping, execution of several versions in parallel and selective mode, upgrading, testing, and version replacement on a remote target system [JKL$^+$04].

6.2.1 Client application provisioning using semantic agent

Semantic agents are implemented as JADE-LEAP agents running on the mobile device and the PC, forming a distributed platform. A *Service offerer agent* is started on the PC, representing a service provider that offers services to the users. A *Service requester agent* represents the user on the mobile device, specifying his requirements for the requested service. They will communicate in order to find the appropriate matching service that best suites user requirements.

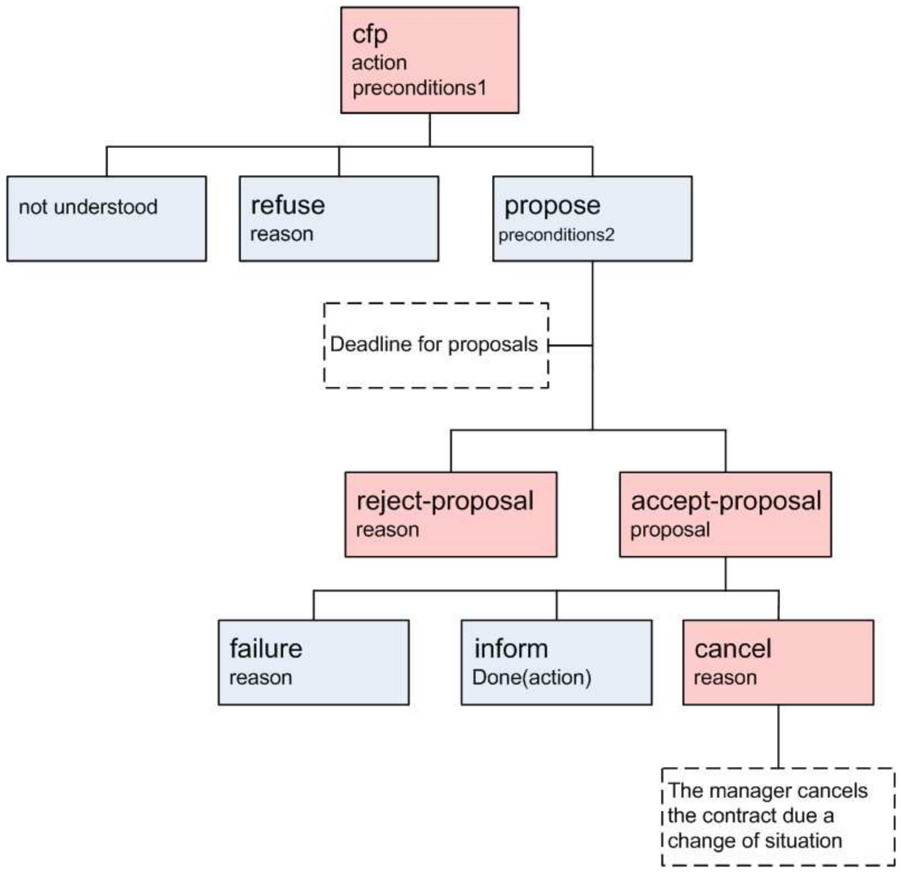
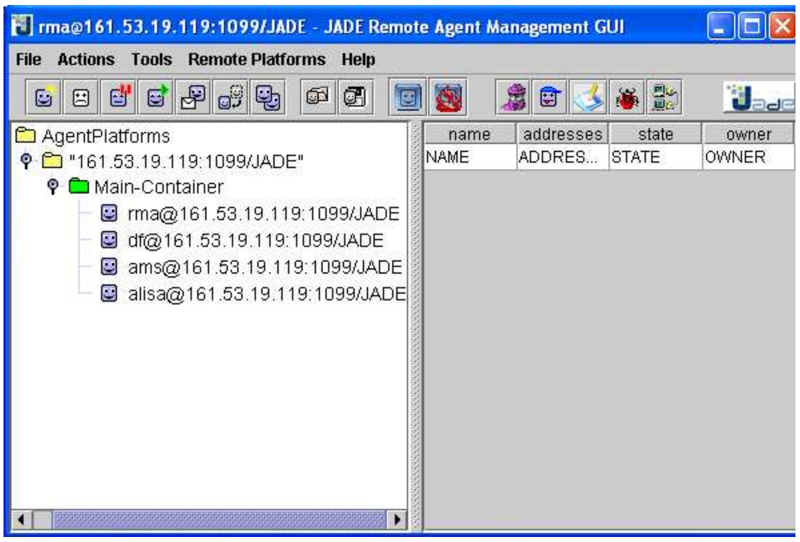


Figure 6.5: FIPA Contract Net interaction protocol

The communication between the service requester and the service offerer agent is realized through the FIPA-Contract-Net protocol. For every conversation among agents, JADE distinguishes an *Initiator* role (the agent initiating the conversation) and a *Responder* role (the agent engaging in a conversation after being contacted by some other agent) [BCTR05]. This interaction protocol allows the Initiator to send a Call for Proposal (CFP) to a set of responders, evaluate their proposals, and then accept the preferred one (or even reject all of them).

The interaction protocol is illustrated in Fig. 6.5, deeply described in FIPA specifications [fip]. The initiator sends a CFP message to obtain proposals from other (responder) agents. The CFP message specifies an action to be performed, and if needed, conditions upon its execution. Responders reply by sending a PROPOSE message including the preconditions that they set out for the action, for instance the price or the time. Alternatively, responders may refuse a proposal by sending a REFUSE message, or eventually, a NOT-UNDERSTOOD to communicate communication problems. The initiator can then evaluate all the received proposals and make its choice of which agent proposals will be accepted and which will be rejected. Once the responders whose proposal has been accepted (i.e. those that have received an ACCEPT-PROPOSAL message) have completed their task, they can, finally, respond with an INFORM of the result of the action (eventually just that the action has been done) or with a failure if anything went wrong.

Before the action has been performed and the last message has been received, the initiator can even decide to cancel the protocol by sending a CANCEL message, but due to the fact that this feature is not well specified in FIPA specifications, it has not yet been implemented in JADE.


rma@161.53.19.119:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help

```
AgentPlatforms
  "161.53.19.119:1099/JADE"
    Main-Container
      rma@161.53.19.119:1099/JADE
      df@161.53.19.119:1099/JADE
      ams@161.53.19.119:1099/JADE
      alisa@161.53.19.119:1099/JADE
```

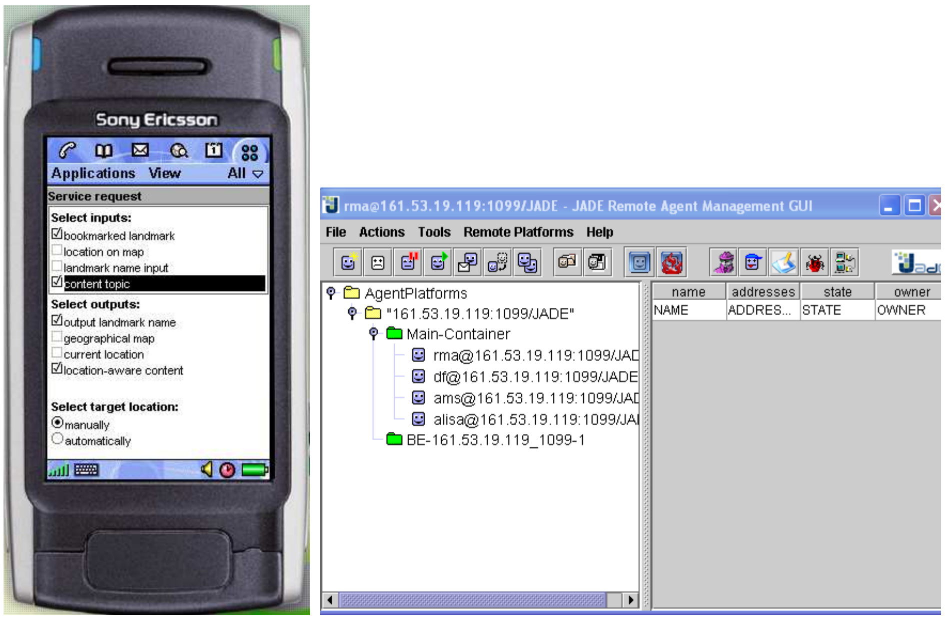
name	addresses	state	owner
NAME	ADDRES...	STATE	OWNER

Figure 6.6: Service offerer agent ("alisa") started in the main container

The service requester agent plays an initiator role while the service offerer agent represents a responder in the conversation. The Figure 6.6 shows the service offerer agent, named *alisa*, started at the J2SE host (PC) in the main container. It is now in the state of waiting for CFP from the service requester agent.

When the service requester agent is created on the mobile device, running a MIDP 1.0 or higher, the midp version of JADE-LEAP is started in a split execution mode. As a consequence, its FrontEnd is added as a BackEnd on the LEAP platform created on the PC (Fig. 6.7).

On the mobile device a graphical user interface (Fig. 6.7) is displayed to the user, where he can select from choice groups inputs, outputs, and a condition that the required service should meet. Inputs and outputs specified in the user interface correspond to inputs and outputs of the atomic processes that are part of the location-aware content delivery service model, described in Section 2.2.1.


Sony Ericsson
Applications View All
rma@161.53.19.119:1099/JADE - JADE Remote Agent Management GUI
File Actions Tools Remote Platforms Help

Service request

Select inputs:
☑ bookmarked landmark
☐ location on map
☐ landmark name input
☑ content topic

Select outputs:
☑ output landmark name
☐ geographical map
☐ current location
☑ location-aware content

Select target location:
● manually
○ automatically

```
AgentPlatforms
  "161.53.19.119:1099/JADE"
    Main-Container
      rma@161.53.19.119:1099/JAD
      df@161.53.19.119:1099/JADE
      ams@161.53.19.119:1099/JAD
      alisa@161.53.19.119:1099/JAI
    BE-161.53.19.119_1099-1
```

name	addresses	state	owner
NAME	ADDRES...	STATE	OWNER

Figure 6.7: Service requester agent started on mobile device

The condition is related to the if-condition associated to the If-Then-Else process, upon which evaluation depends the selection and the execution of one of the two alternative processes.

As the Figure 6.7 illustrates, the user can require a service that uses the *bookmarked landmark* to specify the location where he wants to receive the desired content (i.e. the *content type*). As a result, the service should output the *landmark name* and the *location-aware content* should be delivered to the user's mobile device. The user location should be, therefore, determined *manually* (i.e. by selecting the bookmarked landmark). The selected inputs, outputs, and the condition have to be filled into the CFP message and sent to the service offerer agent.

Application-specific ontologies describe elements that agents use to create the content of messages. Content elements are distinguished by their semantic characteristics into:

- *Predicates* are expressions that say something about the status of the world and can be true or false, e.g.

  ```
  (Works-for (Person :name Alisa) (Company :name FER))
  ```

  stating that "the person Alisa works for the company FER";

- *Terms* are expressions identifying entities (abstract or concrete) that "exist" in the world and that agents talk and reason about. They are further classified into:

- *Concepts* i.e. expressions that indicate entities with a complex structure that can be defined in terms of slots, e.g.

  ```
  (Person :name Alisa :age 25)
  ```

- *Agent actions* i.e. special concepts that indicate actions that can be performed by some agents, e.g.

  ```
  (Sell (Book :title "Harry Potter") (Person :name Alisa))
  ```

- *Primitives* i.e. expressions that indicate atomic entities such as strings and integers;
- *Aggregates* i.e. expressions indicating entities that are groups of other entities e.g.

  ```
  (sequence (Person :name Alisa) (Person :name Alan))
  ```

- *Identifying Referential Expressions (IRE)* i.e. expressions that identify the entity (or entities) for which a given predicate is true e.g.

  ```
  (all ?x (Works-for ?x (Company :name FER))
  ```

  identifying "all the elements x for which the predicate (Works-for ?x (Company :name FER)) is true, i.e. all the persons that work for company FER)." These expressions are typically used in queries and require variables;
- *Variables* i.e. expressions (typically used in queries) that indicate a generic element not known apriori.

An ontology for a given domain is a set of schemas defining the structure of the *predicates*, *agent actions*, and *concepts* that are pertinent to that domain. The appropriate Java classes have to be developed for all type of predicates, agent actions, and concepts and associated to the *PredicateSchema*, *AgentActionSchema*, and *ConceptSchema* classes.

To turn back to the example of the service requester and the service offerer agent, the ontology created consists of the concepts such as: *Service* (used to represent the advertised service) and *Query* (used to specify service requirements), the predicate *Match* (used to match the query against the service), and the agent action *Retrieve* (used to retrieve the service, if the match ended up successfully).

Inputs, outputs and the condition are set into the *Query* concept class, as shown below.

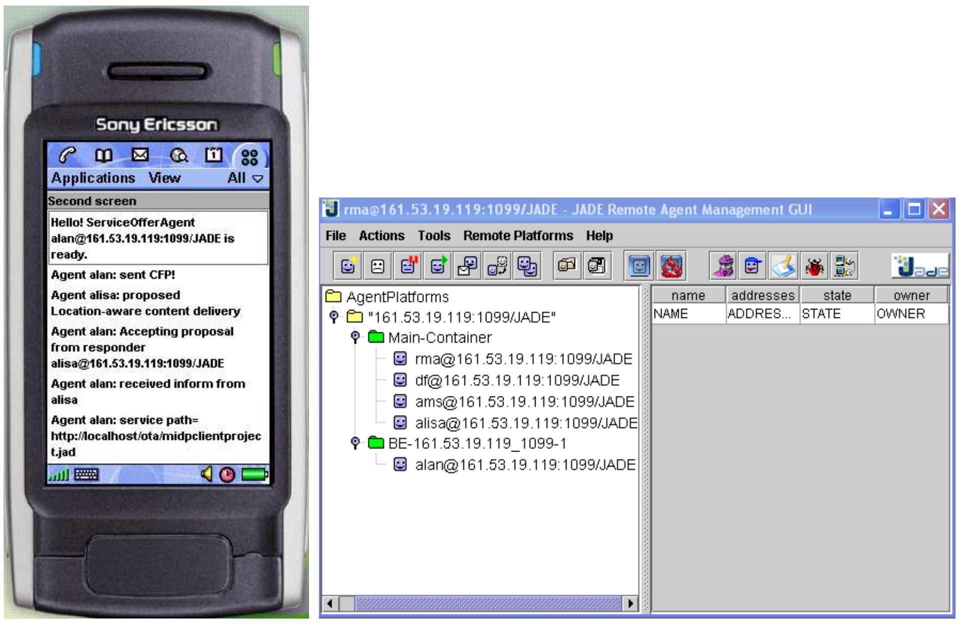
```
public class Query implements Concept{
    private String _inputs;
    private String _outputs;
    private String _condition;
```

```
    public String getCondition() {
        return _condition;
    }

    public String getInputs() {
        return _inputs;
    }

    public String getOutputs() {
        return _outputs;
    }

    public void setCondition(String condition) {
        _condition = condition;
    }

    public void setInputs(String inputs) {
        _inputs = inputs;
    }

    public void setOutputs(String outputs) {
        _outputs = outputs;
    }
}
```

The *Query* is, then, inserted into the *Match* predicate, and finally the *Match* is set as the content object of the CFP message. In order to set the content of a message using the ontology, the ontology and the content language need to be registered with a content manager. In the example, *SLCodec* is used to parse and assembly the content of messages.

```
    Match match = new Match();
    Query query = new Query();
    query.setInputs(inputs);
    query.setOutputs(outputs);
    query.setCondition(condition);
    match.setQuery(query);

    try {
        ACLMessage queryMsg = new ACLMessage(ACLMessage.CFP);
        // Register language and ontology
        queryMsg.setLanguage(FIPANames.ContentLanguage.FIPA_SL0);
        queryMsg.setOntology(ServiceOntology.ONTOLOGY_NAME);
        ...
        getContentManager().fillContent(queryMsg, match);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
```

JADE includes codecs for two content languages: a SL language and a LEAP language. The SL language is a human-readable string-encoded content language and is suggested to agent-based applications to adopt it, because of its openness (i.e. agents developed on different platforms can communicate). On the other hand, the LEAP language is non-human-readable byte-encoded content language that has been defined ah-hoc for JADE within the LEAP project. Although it is lighter than the SL, the LEAP language is intended to be used only by JADE agents.


Sony Ericsson
Applications View All
rma@161.53.19.119:1099/JADE - JADE Remote Agent Management GUI
File Actions Tools Remote Platforms Help

```
Second screen
Hello! ServiceOfferAgent
alan@161.53.19.119:1099/JADE is
ready.
Agent alan: sent CFP!
Agent alisa: proposed
Location-aware content delivery
Agent alan: Accepting proposal
from responder
alisa@161.53.19.119:1099/JADE
Agent alan: received inform from
alisa
Agent alan: service path=
http://localhost/ota/midpclientprojec
t.jad
```

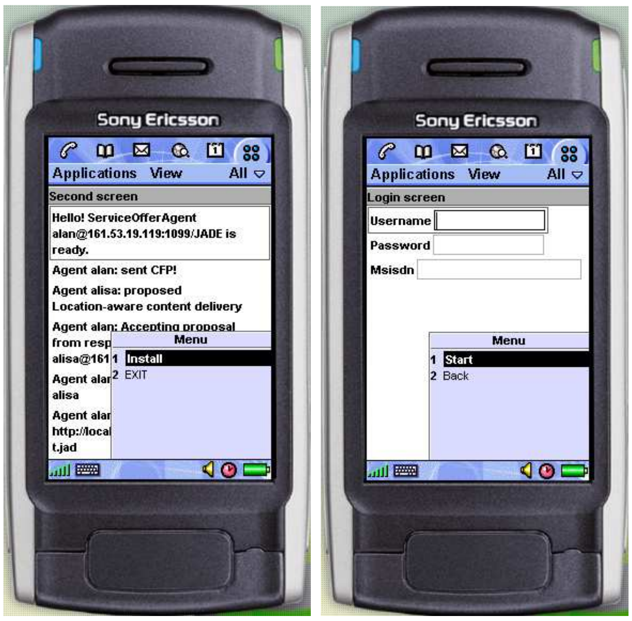
```
AgentPlatforms
  "161.53.19.119:1099/JADE"
    Main-Container
      rma@161.53.19.119:1099/JADE
      df@161.53.19.119:1099/JADE
      ams@161.53.19.119:1099/JADE
      alisa@161.53.19.119:1099/JADE
    BE-161.53.19.119_1099-1
      alan@161.53.19.119:1099/JADE
```

name	addresses	state	owner
NAME	ADDRES...	STATE	OWNER

Figure 6.8: The service output on mobile device

When received the CFP message from the service requester, the service offerer agent parses its content and performs a matchmaking algorithm using retrieved parameters and the ontology advertisement of the service it offers. If the match completes successfully, it sends the PROPOSE message to the service requester agent, containing the name of the service (*Location-aware content delivery service*). Alternatively, if the advertised service does not match user requirements, the REFUSE message is sent to the service requester agent. After receiving the service proposal, the service requester agent replies with the ACCEPT_PROPOSAL; or if the sender does not exist and it didn't received the proposal message from the sender, it replies with the REJECT_PROPOSAL. Finally, if the service offerer agent receives an acceptance of the proposal, it sends the last, INFORM message, in which it inserts a path to the service where it can be retrieved and installed from. More specifically, the service path represents a JAD file of the client application to be installed on the mobile device via OTA provisioning.

The figure 6.8 illustrates the message sequence dialog between the service requester and the service offerer agent outputted on the screen of the mobile device. It also shows the last received message with the path to the service located on the web server on PC. Also, it can be seen on the JADE console gui that the service requester agent, named *alan*, has been created and added to the platform.



Hello! ServiceOfferAgent alan@161.53.19.119:1099/JADE is ready.

Agent alan: sent CFP!

Agent alisa: proposed Location-aware content delivery

Agent alan: Accepting proposal from resp
alisa@161

Agent alar
alisa

Agent alar
http://local
t.jad

Figure 6.9: The service installation and execution on mobile device

The client application provides an ability to install the service on the mobile device. The service installation and execution is shown on Fig. 6.9.

6.2.2 Server-side installation with RMS

The RMS system provides the protected environment for software deployment, upgrade, and testing without suspending or influencing its regular operation. It supports software delivery to the remote system and remote software operations, such as remote installation/uninstallation, start/stop, tracing, maintaining several versions of software, selective or parallel execution of two versions, and version replacement.

Basic RMS concept is shown on Figure 6.10 [JKD$^+$03]. It consists of a management station and of remote systems distributed over the network. The management station is responsible for software delivery to remote systems and for conducting remote operations on them.

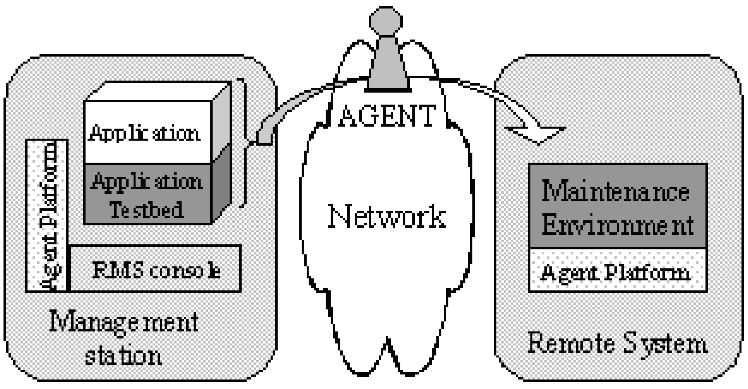


Figure 6.10: RMS concept

A new software (the application) that will be installed using RMS system has to be adapted to RMS. An Application Testbed, the application-dependent part, has to be built along with the application to provide a design for remote maintenance. When the application is ready for delivery, it is migrated together with the Application Testbed on the remote system.

A maintenance environment is the application-independent part of RMS, that is pre-installed on the remote target system(s) to enable maintenance actions. The maintenance environment is responsible for communication with the management station. Its main tasks are enabling remote operations and storing data in the installed software.

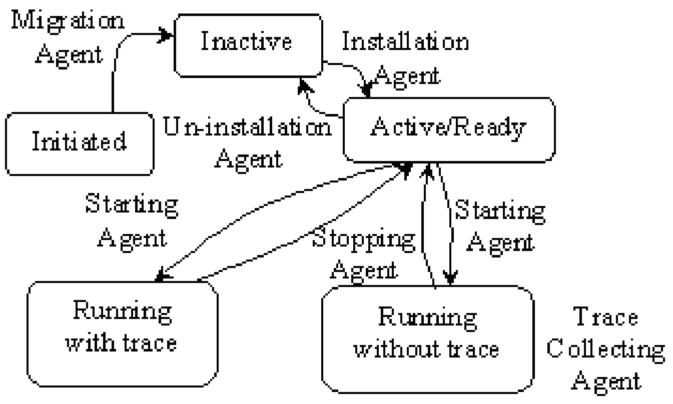
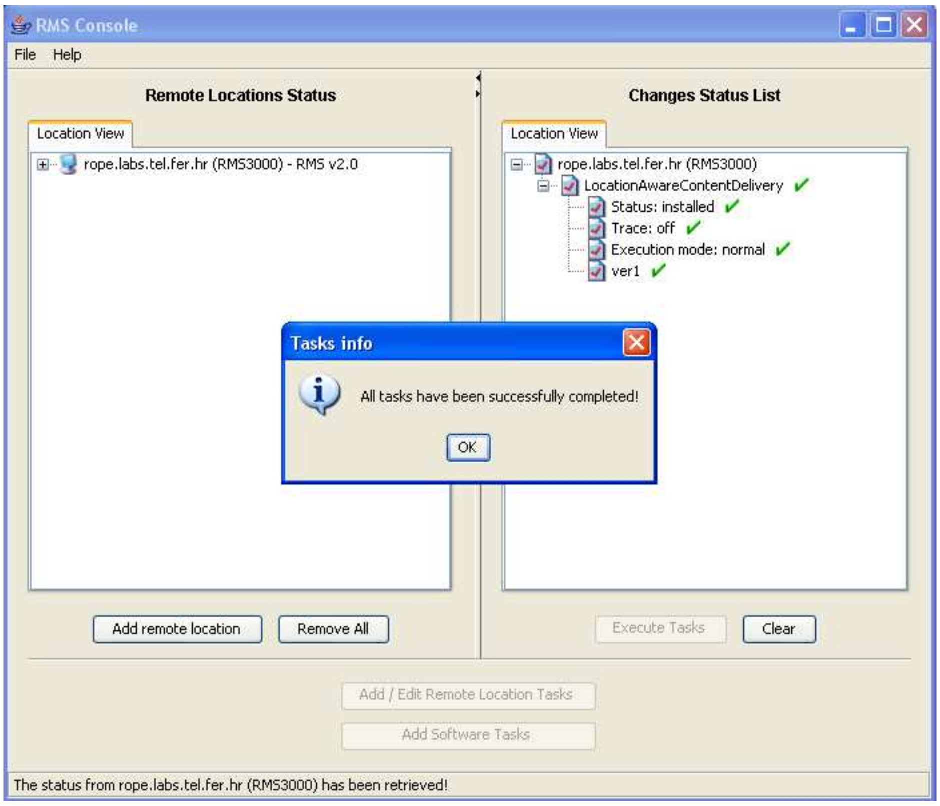


Figure 6.11: Remote maintenance operations with mobile agents

In order to accomplish the given remote operation task, an agent is created and equipped with the required knowledge (communication protocol), data (e.g. software package to be installed), and access rights to the remote system. The



All tasks have been successfully completed!

The status from rope.labs.tel.fer.hr (RMS3000) has been retrieved!

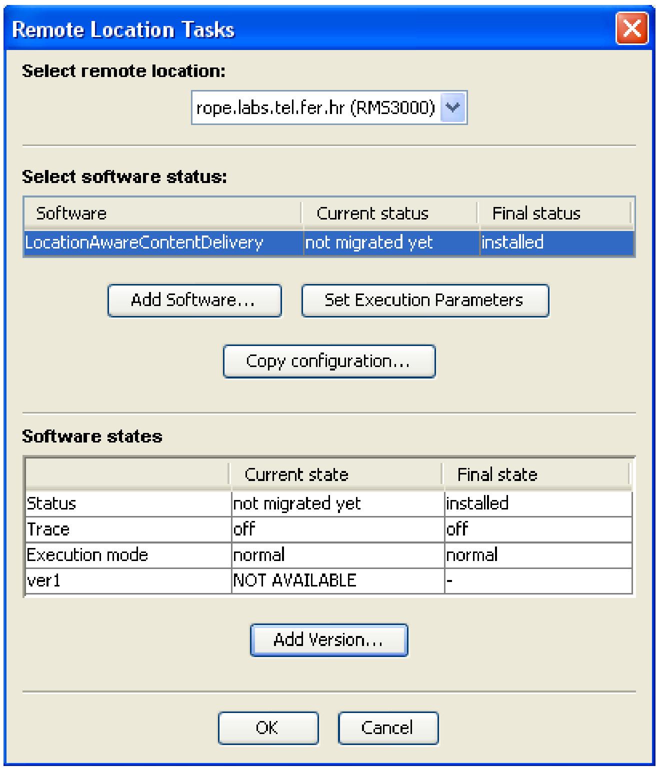
Figure 6.12: Graphical user interface

agent carrying the load is injected into the network and sent to the target system. The same agent can, later, visit other nodes (according to itinerary provided by an agent's owner) and repeat the same action.

Maintenance operations are performed using the set of communicating and co-operating mobile agents, responsible for a particular action. Figure 6.11 illustrates a software state change after completion of a particular operation. Therefore exists one agent for each remote operation: a migration agent responsible for software migration, an installation agent for software installation, an un-installation agent for software un-installation, a starting agent for starting, and a stopping agent for stopping software execution. Only a trace agent, activated by the starting agent, does not change state after it completes the operation. It is used for creating and dispatching the software trace.

Agents are started either directly from a remote console or by other agents. When complete with their actions, agents return back to the management station to report the obtained results.

The server part of the location-aware content delivery service is provisioned us-


Remote Location Tasks

Select remote location:

rope.labs.tel.fer.hr (RMS3000)

Select software status:

Software	Current status	Final status
LocationAwareContentDelivery	not migrated yet	installed

Add Software... Set Execution Parameters

Copy configuration...

Software states

	Current state	Final state
Status	not migrated yet	installed
Trace	off	off
Execution mode	normal	normal
ver1	NOT AVAILABLE	-

Add Version...

OK Cancel

Figure 6.13: Remote Location Tasks dialog

ing RMS system. It is migrated, installed, and run on the remote node (*rope.labs.tel.fer.hr*) using mobile agents in the RMS management console.

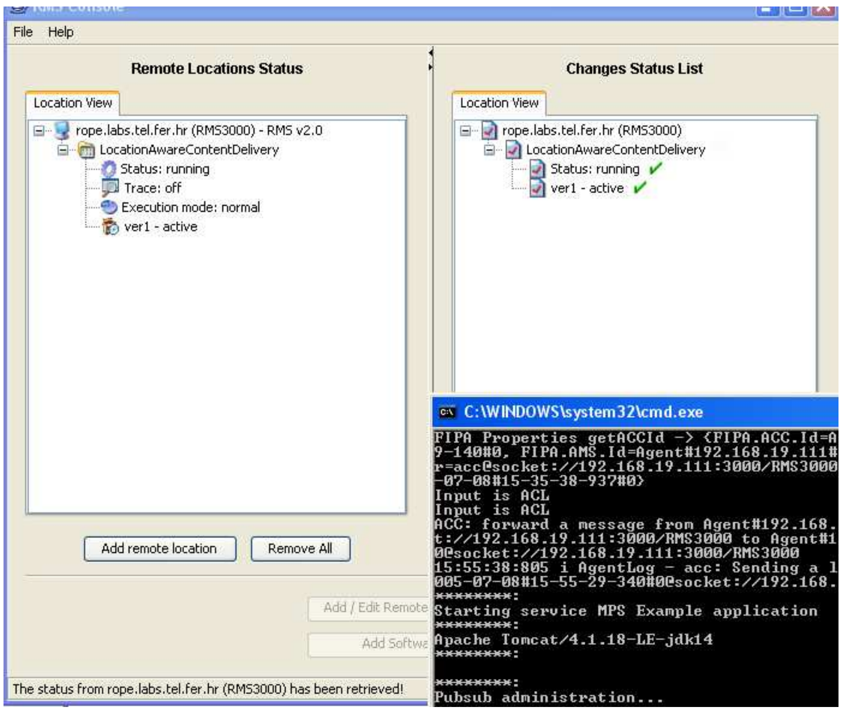
When the console is started, the graphical user interface is displayed on the screen. As it is shown on Fig. 6.12, it is logically separated in two windows: a Remote Locations Status and a Changes Status List.

The Remote Location Status window shows the list of remote locations that are available for managing applications with RMS, and includes the list of software currently available on those locations. The detailed information about the current state of the software managed is available on the screen, as well as the information about the tracing the software and its execution mode. New locations for remote software maintenance can be added, and nodes that are currently monitored can be removed.

The Changes Status List window displays newly assigned tasks that will change the software state on the remote node. Tasks are added through the Add/Edit remote Location Tasks button, specifying the final state of the software on the remote location. When the desired software is selected to be installed on the specified remote locations, tasks can get executed. Changes on remote locations can be easily tracked, since the Changes Status List window is refreshed with green check-marks every time the software parameter reaches the user-defined final state. When all the tasks completed successfully, a confirmation message is popped up in the alert window.

As it can be seen in Fig. 6.12, the remote node *rope.labs.tel.fer.hr* is added to the remote locations list and the location-aware content delivery application testbed, together with the version 1 of the service, is migrated and installed on it. The notification message indicates that all remote operation tasks have been successfully completed.

The Remote Location Tasks dialog (Fig. 6.13) enables the RMS user to add the software that he wants to be installed on the remote location, set its final state (e.g. installed or running), set execution parameters for each software that will be managed on the remote system, and to copy current software configuration to any of remote locations the RMS user is managing. Through the Add Version button user can add a new version of the software, which parameters can be managed on the manner described above.



rope.labs.tel.fer.hr (RMS3000) - RMS v2.0
LocationAwareContentDelivery
Status: running
Trace: off
Execution mode: normal
ver1 - active

rope.labs.tel.fer.hr (RMS3000)
LocationAwareContentDelivery
Status: running ✓
ver1 - active ✓

The status from rope.labs.tel.fer.hr (RMS3000) has been retrieved!

```
FIPA Properties getACCId -> {FIPA.ACC.Id=A
9-140#0, FIPA.AMS.Id=Agent#192.168.19.111#
r=acc@socket://192.168.19.111:3000/RMS3000
-07-08#15-35-38-937#0}
Input is ACL
Input is ACL
ACC: forward a message from Agent#192.168.
t://192.168.19.111:3000/RMS3000 to Agent#1
0@socket://192.168.19.111:3000/RMS3000
15:55:38:805 i AgentLog - acc: Sending a l
005-07-08#15-55-29-340#0@socket://192.168.
*********:
Starting service MPS Example application
*********:
Apache Tomcat/4.1.18-LE-jdk14
*********:

*********:
Pubsub administration...
```

Figure 6.14: Location-aware content delivery service in running state

After the service is installed, it can be executed on the remote node. The figure 6.14 shows the location-aware content delivery service in running state.

Chapter 7

Related work

The section explains different approaches and investigations about the work presented in the thesis. It is logically divided into two parts: first explaining research efforts in the area of the Semantic Web regarding concepts of the semantic services matchmaking using intelligent software agents, and the second presenting the considerable work devoted to location-aware computing and location-based applications.

Semantic services matchmaking. The service matchmaking is the process whereby potential trading partners become aware of each other existence in the e-market [TDJ$^+$05]. A buyer wish to purchase access to a service must find potential service providers able to meet its needs. The matchmaking is required since the buyer's requirements may not be initially fully specified, and service providers may offer similar, but not the same services.

Current standard solutions that are used for matchmaking of Web services, such as WSDL or UDDI, do not support semantic description of services and the matchmaking can only be done by string equality matching on some fields, such as name, location, or URL. To avoid these shortcomings, researchers have been focused on use of web ontologies for providing richer description of service capabilities and finding ways of combining and extending them.

Applying semantic web technologies, specifically RDF and OWL-S (formerly DAML-S), to solve this problem offers a great flexibility and expressiveness to perform service matchmaking. Web services are matched on the basis of the subsumption hierarchy provided by the ontology relating to the concept being matched, rather than on syntactic similarity between them. As such, the ontology provides the context in which services are interpreted.

Approaches that are relevant to the matchmaking algorithm realized in the thesis are the DAML-S Matchmaker based on Service Profile, developed at the Technical University of Berlin [Tan04], and the matchmaker of Web services based on DAML-S Service Model, created at the University of South Carolina [Ban02].

DAML-S Matchmaker. The matchmaking procedure is based on DAML-S Service profile and is split into four stages of matching: an input parameter matching, an output parameter matching, a profile matching, and a user-defined matching. Each stage is independent of the other three, and the final result is based on individual results of these four stages. Degrees of matching will define how well the functionality of any advertised service fits to that of a requested service. The algorithm defines a concept and a property matching, thus classifying different relations between concepts and properties when comparing two concepts or two properties.

The concept matching denotes three types of relationships between the concept A and the concept B:

- *Equivalent(A,B)* - A and B are equivalent, meaning that both denote exactly the same concept.
- *Subsumes(A,B)* - The concept B is subsumed by the concept A, meaning that A denotes a more general concept than B.
- *Fail(A,B)* - Concepts A and B are not in relation to each other with respect to all referenced ontologies.

Similarly, property matching defines three types of relationships between property R and property S:

- *Equivalent(R,S)* - R and S are equivalent which means that both denote exactly the same concept.
- *Subproperty(R,S)* - The property R is a subproperty of the property S.
- *Fail(R,S)* - Two properties R and S are not in relation to each other with respect to all referenced ontologies.

For each parameter (either input or output) several matching degrees of the concept match and the property match exist, depending on the semantic relationship between the service advertisement and request. The property match is given a higher priority than the concept match, since the classification of the parameter gives more insight into its purpose than its type definition. Based on these results a general matching degree is determined.

With the profile matching it can be determined how good the service category of the advertised service fits into the service category that the requested service demands. This matching is based on possibility of classifying the class Profile into subclasses/subcategories. Consequently, the profile matching is based on the concept matching, i.e. the matching between classes.

In the user-defined matching, the entity who manages the matchmaking algorithm can define additional matching functions by the means of plug-in. Effective use of this matching type can be done when evaluating an additional information of the service profile provided by DAML-S, for example exploiting the quality of service aspects for Web services.

The final matching result for the two considered services is composed of the matching results of the four previously described matching parts and of so called minimal matching degrees the user expects. Therefore, the user needs to specify minimal requirements for matching degrees of the input parameter, the output parameter and the profile matching to be satisfied, for the matching algorithm to succeed. In addition, if the user-defined matching fails, then the final matching result will also fail, regardless of results of other three partial matching results.

Simulations of this algorithm have shown the high degree of positive matches, due to the ranking scheme defined for each stage of the matching procedure. What seems as a drawback of the matchmaking algorithm based on Service Profile, is that the user has to define all possible inputs that are expected by the Web service as well as outputs produced by it, in order for the service to be executed.

Matchmaker of Web services based on DAML-S Service Model. The implemented algorithm allows the user to make queries based on the manner in which inputs are transformed into outputs, what is specified by the Service Process Model. It allows the user to differentiate among possible input options, and obtain a successful match by accepting an input corresponding to only one of these options.

The match between the service advertisement and the request occurs when all outputs of the request are matched against outputs of the advertisement and all inputs of the advertisement are matched against the inputs provided by the user needed for the operation. If one of request's outputs is not matched against outputs of the advertisement, the match fails.

Based on the semantic equivalence of the request and the advertisement, the same categories of match have been identified in the algorithm implemented in the thesis:

- *Exact*
- *Plug-In*
- *Subsumes*
- *Fail*

Details about the matchmaking algorithm and a definition of matching degrees are given in Section 2.

Both of the matching algorithms described are based on DAML-S ontologies. The latter algorithm, based on Service Model, is extended and modified in the thesis, to support the matchmaking of Web services descriptions written in OWL-S. OWL-S has a status of a member submission at W3C since the November 2004.

Location-aware computing. Context-aware computing refers to the special capability of the information infrastructure to recognize and react to the real world context. The context, in this sense, includes any number of factors, such as a user identity, his current physical location, weather conditions, the time of the day, date or season, and a user activity (whether the user is asleep or awake, driving, or walking). Location-aware computing systems respond to the user's location, either spontaneously or when activated by a user request. Such a system might utilize the location information without the user's knowledge.

Location-aware computing and location-based services are extremely active areas of research that have important implications for future availability of, and access to, geospatial information. Other examples include the delivery of location-aware information such as notifications of traffic congestion, warnings of severe weather conditions, announcements of nearby educational or cultural events, etc.

Providing the contextualized content to mobile users has gained a lot of attention at this time. Not only third party content providers, but individual users can produce and publish the content. The content can be filtered, personalized, and adapted to certain context elements.

Researchers at VTT Technical Research Centre of Finland have demonstrated the content adaptation based on combined contexts (location, time, social aspects, and device characteristics) on the example of a personalized mobile wap portal in the scope of the KONNTI project [TKL03]. They introduced social contexts consisting of a user's state of mind (or mood), a mode of spending time, and group contexts, as well as provided an RDF serialization of the ontology relevant concepts and their relationships. Mobile users that use this portal need to be always online to access the needed information. The location-aware content delivery service, described in the thesis, offers the means of the content delivery even when the user is offline and is not actively involved in using the application on the mobile device.

The problem of location-based services is that they are mostly bound to a specific positioning technology, and when it happens that the system needs to change it (e.g. from GPS positioning to WLAN or Bluetooth positioning), it has to be entirely reengineered. Authors in [IH04] have described the utilization of different positioning methods in location sensing. They have shown the use of Web services standards (WSDL and SOAP) for propagation, discovery, and composition of location-based services in mobile environments. RDF was used to add semantics to the location information expressed in GPS coordinates or cell ID, addressing its limitations of expressivity and processing capabilities. The location semantics proposed in the thesis offers a solution to integrate different forms of location information in the generic presentation format to be used by various location-aware services.

The GeoNotes system [EPS+01] allows posting notes to other people in the surroundings, sharing context information. Users can participate as content providers and access other people notes depending on preferences, situation, and information needed. On the other hand, the navigation is supported by collecting and aggregating users' usage of the system, and distribute this data to other users in some refined form. Information filtering techniques, therefore, become essential to prevent the information overload and a user's disturbance. Content-based filtering is based on matching the user's keywords. Users' traces are tied to specific geographical positions. Usage-based filters are used to aggregate and match the usage data with other users to enhance and enrich the social navigation and awareness. The system lacks the user mobility functionality and the terminal heterogeneity. The location-aware content delivery system enables the user to apply several terminals to use the same service and to change it during the service use, without suspending its regular operation.

Collaborative work of researchers from Sun Microsystems and University of Lausanne has produced the concept of location-based publish/subscribe system [EGH05], intended for use by mobile ad-hoc applications to communicate with each other based on location. Topics are expressed as dynamic proximity area and there is no possibility for the user to make the subscription to the desired content related to the location of interest. In the thesis the similar architectural pattern of publish/subscribe system as the one created in this thesis has been followed, but for the purpose of location-aware content delivery. The implemented system takes care about tracking the subscriber and filtering the content delivery to the following user preferences: a subscription to the topic depending on his location of interest, a presence information, the utilized terminal, and a preferred delivery method (SMS, MMS, or e-mail).

Conclusion

Current solutions developed in the area of service provisioning and device management in the mobile network greatly involve the Over-The-Air (OTA) provisioning, a standard-accepted and widespread mechanism for remote management of device settings and applications. Instead of users going to the physical location of the service provider to subscribe to the services of interest, the service is automatically delivered and installed on the fly to their mobile device. What lacks is the capability for users to specify their requests about the service they wish to invoke, as well as the ability of the telecommunication system to dynamically discover it, execute and monitor its behavior during the runtime. The next limitation concerns the semantic diversity between the required and the available service(s), since it is highly unlikely that the exact service that matches user requirements will be found.

The role of the Semantic Web and the idea of the semantic matchmaking, performed by intelligent software agents, can be utilized in this purpose. Semantic Web services capabilities are described using OWL-S. Based on the semantic information about the services available, the matchmaking procedure determines the degree of semantic similarity of the requested and advertised services, that can help the user to decide if he wants to make use of the service.

Another benefit of this approach lies in the way the matchmaking algorithm is realized. It is built on service process model that divides service functionality into a set of subprocesses responsible for transforming inputs into the desired outputs. Therefore, it is not necessary for the user to specify all the possible inputs that the service requires to produce the expected outcome.

The thesis has presented a solution for a flexible and efficient service provisioning concept using semantic agents in the mobile network. An agent representing the user requesting the service is started on the user mobile device, and it communicates with an agent acting on behalf of a service provider, offering the available services running on personal computer, that is connected to the Internet. If it happens that after the matchmaking procedure, the appropriate service that matches user demands is found, the user is provided the possibility to install this service over the air on his mobile device.

The novel service that is provisioned this way is the location-aware content delivery service. It delivers personalized content to the mobile subscribers based on their current location, subscription preferences and applied terminal. The content adapted to the device characteristics is sent as SMS, MMS or e-mail to the subscriber's terminal, after it is published on the topic specified by subscriber, and when it is determined that the location of the publisher matches the location specified in the user subscription. The publish/subscribe mechanism is utilized for push-based delivery of data to the subscribers' communication point according to his presence status. At any time the subscriber can set his availability for communication to 'not available' and decide not to receive the time-sensitive information. In that case, the information has to be stored for subsequent delivery until the user communication status becomes 'available' again.

Compared to the existing location-aware systems and content delivery services, this service enables a personalized and selective delivery of the location-aware content to numerous users, using existing transport mechanisms. It is believed that it will open up new excitement opportunities for both the mobile operators and the content providers, offering a variety of information services targeting different users needs.

Literature

[3gp00] 3GPP TS 23.127: Virtual Home Environment/Open Service Architecture. http://www.3gpp.org, 2000.

[3st04] Project: 3store. http://sourceforge.net/projects/threestore/, 2004.

[AvH04] Grigoris Antoniou and Frank van Harmelen. *A Semantic Web Primer*. The MIT Press, Cambridge, Massachusetts, 2004.

[Ban02] Sharad Bansal. *Matchmaking of Web Services based on DAML-S Service Model*. Master thesis, University of South Carolina, 2002.

[BBC97] Peter J. Brown, John D. Bovey, and Xian Chen. Context-aware applications: from the laboratory to the marketspace. *IEEE Personal Communications*, 1997.

[BCM$^+$02] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Application*. Cambridge University Press, 2002.

[BCMS] Fulvio Bosko, Ivano Costa, Corrado Moiso, and Maximiliano Dario Sommantico. 3g service architecture for mobile networks: A reference model. *exp in search of innovation*.

[BCTR05] Fabio Bellifemine, Giovanni Caire, Tiziana Truco, and Giovanni Rimassa. JADE Programmer's Guide. http://jade.tilab.com/doc/programmersguide.pdf, 2005.

[BJR98] Grady Booch, Ivar Jacobson, and James Rumbaugh. *Unified Modelling Language User Guide (Object Technology)*. Addison-Wesley, 1998.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 2001.

[Cai05] Giovanni Caire. LEAP User Guide. http://jade.tilab.com/doc/LEAPUserGuide.pdf, 2005.

[CBB+02] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures*. Addison-Wesley, 2002.

[CCP03] Composite Capabilities/preference profiles (CC/PP): Structure and vocabularies. W3C Working Draft available at: http://www.w3c.org/TR/CCPP-struct-vocab/, March 2003.

[CK00] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical report TR2000-381, 2000.

[dam00] DAML-ONT Initial Release. http://www.daml.org/2000/10/daml-ont.html, 2000.

[dam01] DAML+OIL. http://www.daml.org/2001/03/daml+oil-index.html, 2001.

[DJ05] Alisa Devlic and Gordan Jezic. Location-Aware Information Services using User Profile Matching. In *Proceedings of the 8th International Conference on Telecommunications (ConTEL 2005)*, pages 327–334. University of Zagreb, Faculty of Electrical Engineering and Computing, 2005.

[DP03] Alisa Devlic and Ivana Podnar. Location-Aware Content Delivery Service Using Publish/Subscribe. In *In Proceedings of the Telecommunications and Mobile Computing (tcmc 2003)*, pages 159–177. OVE-Mediacenter, Austrian Electrotechnical Association (OVE), 2003.

[EGH05] Patrick Th. Eugster, Benoit Garbinato, and Adrian Holzer. Location-based Publish/Subscribe. UNIL Technical report DOP-20050124, 2005.

[EH03] E.J.Freiedman-Hill. Jess in Action: Java Rule-based System. Manning Publications, Greenwich, Connecticut, US, 2003.

[EPS$^+$01] Fredrik Espinoza, Per Persson, Anna Sandin, Hanna Nyström, Elenor Cacciatore, and Markus Bylund. Geonotes: Social and navigational aspects of location-based information systems. In *Proceedings of the Ubiquitous Computing*, 2001.

[eur01] MobilUS: Next generation Mobile Information Services on UMTS. http://www.eurescom.de/ pub-deliverables/P1100-series/P1105/TI1/p1105ti1.pdf,
2001.

[fip] The Foundation for Intelligent Physical Agents (FIPA). http://www.fipa.org.

[Gua98] N. Guarino. Formal Ontology in Information Systems. In *Proceedings of the Formal Ontology in Information Systems (FOIS 1998)*, pages 3–15. Amsterdam IOS Press, 1998.

[HBS04] Albert Held, Sven Buchholz, and Alexander Schill. Modelling of Context Information for Pervasive Computing Applications. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics (SCI 2004)*, 2004.

[Hen01] James Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16:30–37, 2001.

[Her96] Björn Hermans. Intelligent Software Agents on the Internet. http://www.broadcatch.com/agent_thesis/, 1996.

[HP] HP Labs Semantic Web research. http://www.hpl.hp.com/semweb/.

[IH04] Peter Ibach and Matthias Horbank. Highly available location-based services in mobile environments. In *Proceedings of the International Service Availability Symposium*, 2004.

[jad] Java Agent Development Framework (JADE). http://jade.tilab.com.

[Jen] Jena 2 - A Semantic Web Framework. http://www.hpl.hp.com/semweb/jena2.htm.

[JKD$^+$03] GordanJežić, Mario Kušek, Saša Dešić, Antun Carić, and Darko Huljenić. Multi-Agent System for Remote Software Operations. In

Knowledge-Based Intelligent Information and Engineering Systems (ISSN 0302-9743), volume 2774. Lecture Notes in Computer Science, Springer-Verlag, 2003.

[JKL^{+}04] Gordan Ježić, Mario Kušek, Ignac Lovrek, Saša Dešić, and Björn Dellas. Agent-based framework for distributed service management. In *Proceedings of the 16th IASTED International Conferance on Parallel and Distributed Computing and Systems*, pages 583–588. Cambridge, ACTA Press, 2004.

[jor] Java Open Reliable Assynchronous Messaging (JORAM). http://joram.objectweb.org/.

[jos03] Joseki - The Jena RDF Server. http://www.joseki.org/, 2003.

[Knu03] Jonathan Knudsen. *Wireless Java: Developing with J2ME.* Apress, 2003.

[Lov99] Ignac Lovrek. Soft Mobility, 1999.

[MDT05] Ivan Mećar, Alisa Devlić, and Krunoslav Tržec. Agent-oriented Semantic Discovery and Matchmaking of Web Services. In *Proceedings of the 8th International Conference on Telecommunications (ConTEL 2005)*, pages 603–607. University of Zagreb, Faculty of Electrical Engineering and Computing, 2005.

[NAW94] Bill N.Schilit, Norman Adams, and Roy Want. Context-aware computing applications. In *Proceedings of the First IEEE workshop on Mobile Computing Systems and Applications*, 1994.

[oil00] Description of OIL. http://www.ontoknowledge.org/oil/, 2000.

[Ort02] C. Enrique Ortiz. Introduction to OTA application provisioning. http://developers.sun.com/techtopics/mobility/midp/articles/ota/, 2002.

[ota01] Over The Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile. http://java.sun.com/products/midp/OTAProvisioning-1.0.pdf, 2001.

[ota02] MIDP 2.0 specification. http://www.jcp.org/jsr/detail/124.jsp, 2002.

[owl] OWLJessKB: A Semantic Web Reasoning Tool.
http://edge.cs.drexel.edu/assemblies/software/owljesskb.

[owl02] Feature Synopsis for OWL Lite and OWL.
http://www.w3.org/TR/2002/WD-owl-features-20020729/, 2002.

[owl04] OWL Web Ontology Language Overview.
http://www.w3.org/TR/owl-features/, 2004.

[php] PHP XML Classes. http://phpxmlclasses.sourceforge.net/.

[pos01] Cellular location technology.
http://www.vtt.fi/tte/tte35/pdfs/CELLO-WP2-VTT-D03-007-
Int.pdf, November
2001.

[rap04] RAP - Rdf API for PHP.
http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/, 2004.

[RDF] RDF Test Cases. http://www.w3.org/TR/rdf-testcases/.

[RDF03] RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft available at: http://www.w3c.org/TR/rdf-schema/, January 2003.

[rdf04a] RDF Vocabulary Description Language 1.0: RDF Schema.
http://www.w3.org/TR/rdf-schema, 2004.

[rdf04b] RDFStore - PERL/C RDF storage and API.
http://rdfstore.sourceforge.net/, 2004.

[rdf04c] RDF/XML Syntax Specification (Revised).
http://www.w3.org/TR/rdf-syntax-grammar, 2004.

[rdq04] RDQL - A Query Language for RDF.
http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/,
2004.

[RLF00] Andry Rakotonirainy, Seng Wai Loke, and Geraldine Fitzpatric. Context-awareness for the mobile environment. In *Proceedings of the Human Factors in Computing Systems Conference*, 2000.

[ses05] Sesame: RDF Schema Querying and Storage. http://www.openrdf.org/, 2005.

[Tan04] Stefan Tang. *Matching of Web Service Specifications using DAML-S Descriptions*. Diploma Thesis, Technical University of Berlin, 2004.

[TDJ$^+$04] Krunoslav Trzec, Alisa Devlic, Gordan Jezic, Mario Kusek, and Sasa Desic. Semantic Matchmaking of Advanced Personalized Mobile Services using Intelligent Agents. In *In Proceedings of the 12th International Conference on Software, Telecommunications and Computer Networks (SoftCOM 2004)*, pages 387–391. University of Split, Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture, 2004.

[TDJ$^+$05] Krunoslav Tržec, Alisa Devlić, Gordan Ježić, Mario Kušek, and Saša Dešić. Semantic matchmaking of mobile web services using intelligent agents. *Journal of Communication Software and Systems (JCOMSS)*, to be published in 2005.

[tel03] Context-aware services. https://doc.telin.nl/dscgi/ds.py/Get/File-27859/Context-aware_services-sota,_v3.0,_final.pdf, November 2003.

[TKL03] Sanntu Toivonen, Juha Kolari, and Timo Laakko. Facilitating mobile users with contextualized content. In *Proceedings of the Artificial Intelligence in Mobile System Workshop (AIMS 2003)*, 2003.

[UaP99] Wireless Application Group User Agent Profile Specification. http://www.wapforum.org/what/technical/SPEC-UAProf-19991110.pdf,
1999.

[wvi02] Wireless Village Presence Attributes V1.0. http://www.openmobilealliance.org/tech/affiliates/wv/wv_pa_-v1.0.pdf,
2002.

[xpa99] XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath, 1999.

Summary

The master thesis investigates the issues of service provisioning in mobile network. Service provisioning is defined as the setting in place and configuring of the hardware and software required for activating a telecommunications service for a customer. The thesis proposes and implements a solution for a flexible and efficient service provisioning using semantic agents in the mobile network. Semantic agents are intelligent software agents that collect user preferences for the required service, and discover the available advertised services offered by service providers. They use the implemented matchmaking algorithm in the thesis to determine from the available services the one that best meets user's requirements. When the service is determined and found on the Web, the user can install and invoke it on the mobile device.

The service that was utilized for service provisioning is the location-aware content delivery service that delivers personalized content to mobile users depending on their current location, utilized terminal and preferences. It consists of the client and server part, for which the provisioning is performed separately. The client part is provisioned on the described manner, using semantic agents. The server part of the service is migrated, installed and invoked on the remote node, using the multi-agent system supporting remote software maintenance operations in the network.

Keywords:
semantic agents, location-aware services, personalized content delivery, service provisioning, semantic matchmaking

Curriculum Vitae

I was born on August 15th, 1979 in Kutina. After finishing high school with excellent grades, I started the undergraduate program at the Faculty of Electrical Engineering and Computing, University of Zagreb, in 1997. I graduated on May, 15th 2002 from the Telecommunication Department. My Diploma thesis was based on location-based services and developing an application based on mobile positioning for Palm OS mobile devices in GSM/GPRS environment. I am currently a guest researcher at Wireless@KTH center in Kista, Sweden, where I am involved in the "Adaptive & Context-Aware Services" research project, under the guidance of prof. Maguire from the KTH. I was employed as a research associate with the Telecommunication Department, FER, from July 2002 till September 2005, when I have been granted a scholarship from the Swedish Institute. I was involved in the research project Remote Operation Management, under the leadership of Gordan Ježić, that is performed in cooperation with Ericsson Nikola Tesla company. My current research interests include location and context-aware services, mobile software agents, as well as issues of user, terminal and service mobility in mobile networks. I have published 6 papers on international and domestic conferences in the area of location-based services and mobile agents. I am fluent in English and German. I am a member of IEEE.