Master's Thesis

A Case Study and Experimental Evaluation of Web Services in Mobile Computing

carried out at the

Information Systems Institute
Distributed Systems Group
Vienna University of Technology

under the guidance of
o.Univ.Prof. Dr. Schahram Dustdar
as the contributing advisor responsible

by

Bernhard Wehinger
Dammstrasse 6, A-2100 Korneuburg
Matr.Nr. 9926255

Vienna, 21. February 2008 ______________________

Danksagung

Ich bedanke mich bei meiner Familie, die mir während der nicht immer ganz einfachen Studienzeit unablässig zur Seite gestanden ist.

Besonderer Dank gilt dabei meinen Eltern, die mir durch finanzielle und moralische Unterstützung dieses Studium ermöglicht haben.

Ich danke auch Daniel Schall für die professionelle Betreuung während der Entstehung dieser Arbeit.

Abstract

Today more and more people make use of mobile digital assistants and Smartphones, as the capabilites and performance of those devices are getting improved constantly. Through the expansion of the infrastructure for mobile internet, the devices can go online nearly everywhere and at any time. So why not make use of all these devices that have enough computing power and resources to potentially act as servers? The goal of this work is to evaluate the feasibility of an OSGi based context-aware collaboration framework on lightweight devices like Pocket PCs and mobile phones running Java for mobile devices. The framework developed in the course of this thesis aims at supporting collaboration in ad-hoc and mobile teams. The assumption is that no centralized infrastructure exists, thus collocated devices and services need to be discovered in ad-hoc mode. Particular attention will be given at standard Web services, hosted on mobile devices, as communication means between devices and a mobile Context Store for context information retrieval and aggregation.

Zusammenfassung

Mobile digitale Helfer wie Pocket PCs und Smartphones haben in vielen Bereichen unseres Lebens Einzug genommen, da diese Geräte immer schneller und ihre Möglichkeiten ständig erweitert werden. Durch die Erweiterung der Infrastruktur für mobiles Internet, kann man nahezu immer und überall online gehen. Es liegt deshalb nahe, mobile Geräte, die über genügend Rechenkapazität und Resourcen verfügen, als Server für diverse Services zu nutzen. Ziel dieser Arbeit ist es, Context Informationen auf einem mobilen Gerät zur Verfügung zu stellen. Das entwickelte Framework basiert auf OSGi und läuft auf allen Geräten die J2ME/CDC mit Personal Profile unterstützen. Weiters wird angenommen, dass keine zentralisierte Infrastruktur zur Verfügung steht, was einen Discovery Mechanismus für Geräte und Services erfordert. Besondere Aufmerksamkeit wird dabei Standard Web Services gewidmet, die auf den mobilen Geräten gehostet werden müssen und als Kommunikationsmethode dienen.

Contents

List of Figures

List of Tables

1 Introduction

The variety of mobile devices is increasing rapidly and the technical infrastructure for communication and data transfer is constantly improved and expanded nowadays. Thus there is a substantial interest in leveraging this progress of the communication and information technology to support collaborative scenarios.

This work is done as a part of the inContext project, which aims at developing a toolkit to support the creation of collaborative services with focus on mobility and teams. Due to the advancements of mobile and wireless technologies, people, who rely on mobile devices like laptops, Smartphones etc., are more flexible in choosing their working environment and can use the tools they need nearly everywhere. Mobility and heterogeneity of devices cause connectivity, availability and bandwidth issues [33].

To eliminate some of these issues from the outset, this work concentrates on standard Web services to take advantage of the platform independent nature of Web services, which allow applications of all kinds to interact with each other [36].

1.1 Motivation

The aim of this work is to allow people and teams to query and share context information using off-the-shelf mobile devices. Being able to share context using mobile devices enables people to benefit from context sensitive applications anywhere and not only when a connection to fixed devices, which provide context information, is established. In a mobile environment with constantly moving devices, a software architecture supporting mobile collaboration has to ensure that the devices are aware of each other. The foundation of this work is a discovery mechanism for making the mobile devices aware of each other and to exchange context data using a Web service provider on a lightweight device.

1.1.1 Mobile / Ad-hoc Teams in Collaborative Environments

Most companies organize their employees in teams and each of them is contributing to a common goal. When a team needs expert knowledge from one or more specialists of other companies, who are maybe located in a foreign country or different time zone, the workflow, communication, data exchange etc. will become much more complex. Nimble teams emerge [37], which are a special form of a collaboration teams that are tightly coupled, short-lived and very flexible in their team configuration. An example of a nimble team is a taskforce. Because of the pervasiveness of mobile devices, the ability to connect to the internet nearly always and everywhere and the advanced integration into our daily life, those teams are no longer bound to one location. Flexible working schedules and the fact, that people are often engaged in more than one task or project at the same time require new context-aware means of communication and collaboration to be developed.

As one can imagine, experts of companies often have to switch between the teams they are consulting. Therefore a very important requirement for any software supporting human workflows is to keep track of the context of the user. Consider for example a car manufacturing company having problems adjusting the control unit for an antilock brake system. The knowledge of an expert of the company producing this control unit is needed to assist the engineering team solving the problem. But most likely this unit is sold to other car makers as well and they maybe have some similar troubles using it, so the same specialist has to support many teams virtually at the same time. Therefore software services supporting the specialist have to "know" which team he is part of, where he is located, which device he is using and on which task he is working at the moment to provide relevant information like status of the other team members, documents, files etc. Context information is vital to effectively support knowledge workers in their workflow and helps workers to select and provide relevant information to other team members.

Due to the importance of context in nimble/mobile teams, it is necessary to investigate a way to enable team members to search for information. There are two different types of search queries.

First, the system should enable people to query the context of coworkers. For example it could be important to know, where the other team members are located or which status they have at the moment. Consider a team leader planning a meeting to discuss future topics of the project. He has to check the availability of his colleagues and has to reflect about whom he is inviting. Imagine that there are some important specialists whose attendance at the meeting is absolutely necessary. In this case it is of paramount importance for the team leader to know the location of his specialists or if they are currently engaged in some other temporally critical activities and need not be disturbed. This information helps the team leader to choose the right time and the right place for the meeting.

If it is not possible to arrange a meeting because team members are indespensable, the team leader could decide to switch to a videoconference instead of a personal meeting. If we take a look at this case, we can see, that many challenges arise here. Several prerequisites must be satisfied to successfully establish a videoconference. The team leader or the system has to know, which mobile device the team members are currently using e.g. Smartphone, Pocket PC or notebook and whether all of these devices fulfill the needs of a video conference. Furthermore, the network bandwidth of the remote devices plays an important role, as the video conferencing system might need to stronger compress the video stream for proper presentation on a Pocket PC with UMTS than on a laptop with T1 connection and there are many more aspects of team collaboration scenarios, that show the importance of context of users and devices to increase the efficiency of teamwork.

Besides the ability to search for context details, project workers must also be able to lookup information concerning the current project or only a task of the team, for example files, documents, meeting protocols, activity history of team members etc.

Finally the combination of the two search domains (context information and artefacts concerning the currently ongoing project) empowers the user to execute very useful queries. For acquiring these details, a mechanism must be provided to query mobile devices.

1.2 Problem Description

Using toolkits and frameworks available today, it is possible to consume Web services from mobile devices. For example kSOAP is a SOAP library with a small footprint which can be used on mobile devices running Java-based applications to communicate with SOAP based Web services. Also the .NET Compact Framework from Microsoft offers the possibility to integrate SOAP based Web services seamlessly into .NET compact based programs.

But the situation is different when it comes to providing Web services on a mobile device. There exist only few Web service toolkits for mobile and especially lightweight devices. Popular Web service toolkits like Apache Axis are way to heavyweight to run on a mobile device with limited resources. Although there are editions available where only the core components are included, these toolkits cannot be used on Smartphones or PDAs. In this work possibilities of providing Web services on lighweight mobile devices will be investigated.

The goal of this work is to demonstrate the feasibility of a lightweight, platform-independent framework for providing context information on devices with limited hard- and software capabilities. The proof-of-concept implementation as presented in this thesis in Section 4 aims at demonstrating the feasibility of an OSGi-based framework, which acts as hosting environment for a Context Store and Web services to allow communication between devices using standard protocols (i.e. SOAP). The evaluation will show, if the limited nature of mobile devices (bandwidth, performance, memory etc.) allows the provision of Web services in a reasonable manner or if it is even possible.

Since it cannot always be assumed that mobile devices are connected to a network providing a centralized infrastructure, this thesis follows the paradigm of a decentralized architecture, which requires a service discovery mechanism. Due to the strong mobility, loose coupling is necessary, because it is most likely that mobile devices with wireless connection move out of network coverage from time to time. This work also describes current service discovery and registration mechanisms.

1.3 Contributions

1.3.1 Evaluation of Web Services on Mobile Devices

Different Web services technology options currently exist to design and develop Web services-based applications for mobile devices. This thesis provides a detailed comparison of different Web services toolkits. We will take a look at the toolkit features, but also on the advantages and drawbacks of the tools such as deployment options on mobile devices. The comparison mainly focuses on Web service providers, which are available for the mobile Java platform.

Furthermore, this work analyzes the deployment and run-time requirements of Web services on mobile devices, starting with an introduction to mobile devices frameworks and an analysis of their suitability for Web service deployment. Finally, a detailed guideline for deploying Web services using OSGi with the IBM J9 JVM including libraries and configurations will be given.

1.3.2 Architecture for Mobile Collaboration using Web Services

Based on the discussion of different Web service toolkits, we introduce an architecture for mobile collaboration using Web services. The proposed architecture and system allows Web services to be designed, deployed, and managed in a mobile computing infrastructure. The architecture is based on the SOA paradigm and allows services to be registered, discovered, and invoked in a distributed environment. Our architecture supports both ad-hoc and infrastructure mode in registering and discovering services.

1.4 Overview of Thesis

This thesis is organized as follows: Section 2 introduces to mobile devices and their limited capabilities, outlines mobile devices frameworks available today and their suitability to host Web services. This section also gives an overview about the REST architectural style and the JXTA P2P framework. Section 3 deals with the SOA paradigm, technologies and toolkits for creating Web services and their deployment and presents a detailed discussion of service discovery protocols. Section 4 describes the architecture for mobile collaboration developed during this work. In section 5 the reader can find the results of the evaluation of the proof-of-concept implementation. The appendix includes additional information on the prototype.

2 Background

2.1 Interaction Styles

In this thesis particular attention is given on both roles in a service-oriented system: the service consumer and the service provider. Current specifications and implementations of mobile Web services for the Java platform such as JSR 172 [15] only support the design and development of Web services consumer on mobile devices. However, JSR 172 does not address how to host Web services (service provider role) on mobile devices. As highlighted in the motivation example, in the framework it is equally important to be able to consume and also to provide Web services on mobile devices.

Convential Web services mainly utilize the Simple Object Access Protocol (SOAP). SOAP is a message format based on XML, which makes the message creation and parsing calculation intensive. Sangyoon Oh presents an interesting approach to cope with this problem with his Handheld Flexible Representation. Using this framework, applications are able to negotiate the message and streamformat for communication. Performance gain opposite to conventional SOAP based Web services is reached through a binary encoding of the messages. This addresses the slow performance of mobile devices, and the limited availability of high speed networks.

Mobile Web services are difficult to deploy, because of the limitations of the devices like memory, processing power, bandwidth. However, this work investigates, if conventional SOAP based Web services developed with toolkits available today deliver useful results on current average mobile devices.

2.1.1 HTTP

The Hypertext Transfer Protocol [9] is the main protocol used on the Internet to transfer information between clients and servers and originated from the need of a way to publish and retrieve HTML (HyperText Markup Language) documents. HTTP is a request/response protocol where typically the client (often referred to as user agent) sends a request to a server using a transmission control protocol connection to the remote port 80 (default).

Example request:

```
GET /information/index.html HTTP/1.1
Host: www.remotehost.net
```

In this request the user agent challenges the server `www.remotehost.net` to send him the HTML document `index.html` in the directory `information`. `HTTP/1.1` specifies the protocol version. This is the most basic version of a HTTP request, it can be specified more detailed using additional fields in the header of the request (e.g. accept-language, accept-encoding, etc.).

If the document is found, the server answers with a positive response. If the document is not found, the server responds with the corresponding status code (HTTP/1.1 404 Not found). Many other responses are possible, but listing all of them would be beyond of the scope of this work.

Example of a positive response:

```
HTTP/1.1 200 OK
Date: Tue, 25 September 2007 12:31:16 GMT
Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux)
Accept-Ranges: bytes
Content-Length: 1255
Connection: close
Content-Type: text/html; charset=UTF-8

<html> [...] </html>
```

The header tells the client, that the document was found. The content-type identifies the document as a HTML document, which is encoded as UTF-8 and 1255 bytes (accept-ranges) long (content-length). Once the document is received, the user agent can process the document (e.g. a browser will display the file, a search engine robot will parse and index it, etc.).

2.1.2 REST

The term REST was first introduced in Roy Thomas Fieldings PhD thesis and stands for REpresentational State Transfer [38]. REST is not a technology, but an architectural style and a different approach to realize service oriented or web oriented architectures. REST basically makes use of the HTTP protocol, which is used to operate on resources identified by URIs (Uniform Resource Identifiers).

- GET (Retrieves a resource, which can be either data or a URI to another resource)

- POST (Modifies details of a resource on the server)
- PUT (Creates a new resource on the server)
- DELETE (Removes a resource from the server)

Due to the usage of HTTP, REST is a rather lightweight possibility to create service oriented applications. The message format can be chosen freely, but the design principles of HTTP can lead to some problems. HTTP is a stateless protocol, which requires the client to submit all information the server needs to fulfill the request. It is also not possible to implement transactions and do a rollback in case of a failure. The synchronous communication of HTTP can lead to timeouts, if an operation takes too much time.

2.1.3 SOAP

SOAP (Simple Object Access Protocol) is a protocol standard of the W3C [24], which allows distributed (web) applications and objects to communicate with each other. The SOAP messages are XML-based and contain information like namespaces, endpoint, remote procedure and parameters to be used. It can therefore be called a RPC mechanism.

SOAP is platform and language independent and can be used with various transport protocols (binding) like HTTP and SMTP/POP3. The most common case is to use HTTP, because this standard protocol is sophisticated and in most cases firewalls are configured to pass through HTTP requests, which allows clients to operate almost anywhere.

As with all XML-based protocol the biggest trade-off is the low performance due to the large overhead caused by the markup language.

Take a look at the appendix for SOAP messages produced by prototype developed during this work.

2.1.4 JXTA

JXTA is an open source P2P protocol specification originally conceived by Sun Microsystems [44]. It consists of a set of XML messages and was designed to overcome the shortcomings of existing peer-to-peer systems. An example for that would be a music filesharing system based on the peer-to-peer technology, which has the disadvantage, that it is intended for one purpose only. JXTA is intended to be more versatile and applicable in many fields of distributed computing.

Applications based on JXTA are set atop a small and thin layer, which provides powerful primitives and services the application can make use of. Two key features of JXTA are platform independence and ubiquity, which enables it to run on nearly every device and operating system available.

JXTA is defined as a set of protocols and the technology is not based on APIs, which different programming languages and heterogenous devices running completely different software stacks greatly benefit of. Also the choice of the transport protocol itself is up to the developer. Besides TCP/IP, HTTP, Bluetooth, many others are possible options.

2.2 Mobile Devices

2.2.1 Limited Capabilities

Mobile devices are constraint in their processing power and rely on batteries with limited capacity and thus require a careful design of mobile applications to optimize resource consumption such as CPU cycles and memory usage. Limited hardware capabilities and constraint resources of mobile devices lead to software libraries and mobile computing frameworks that are tailored to the needs of those devices. For example, frameworks available for mobile platforms offer libraries to access wireless network interfaces and to design user interfaces that are suitable for representing an application on a mobile screen. On the other hand, some libraries (e.g., XML libraries) are not available on mobile devices, thus making it challenging to reuse existing applications and tools that were developed for PCs and desktops. As we will see in Section 3 Apache Axis is an example for that issue.

2.2.2 Mobile Devices Frameworks

The .NET Compact Framework

In January 2002 Microsoft announced the first release of the .NET Framework. The .NET platform is based on the Common Language Infrastructure (CLI), which enables access to the Framework Class Library (FCL) with multiple languages, e.g. C#, VB.net, C++, J#.

.NET is theoretically platform independent, but Microsoft certainly provides the platform only with its own operating system Windows in all facets. Due to the limited nature of pocket devices, Microsoft has created the .NET Compact Framework, which is a modified and more lightweight variant of the platform for its mobile operating systems. In Windows Mobile 2003 SE version 1.0 of the .NET Framework is already integrated in the ROM of the Pocket PCs and Smartphones. Still, it is upgradable to version 2.0, but it has to be installed in the working memory of the Pocket PC. Since Windows Mobile 2003 SE is rather outdated these days, it is only deployed on older devices. The upgrade to version 2.0 is therefore a rather big trade off to make, because it consumes a considerable amount of internal memory. Windows Mobile 5.0 yet incorporates the new version of the .NET Framework.

As mentioned above, the capacities of pocket devices are very limited. Therefore the .NET Compact Framework does not provide all of the services and functions as the full version. In fact, the size of the redistributable in version 1.0 was shrunk to around 12% of the size of the full platform. Figure 1 shows a comparison between the .NET Framework and the .NET Compact Framework [5]. Modules marked red are available, yellow means partially available and the grey modules are not available in the compact distribution.

The .NET Compact Framework in combination with the Microsoft Visual Studio IDE offers a very simple way to consume Web services in mobile applications. In this project Web services with SOAP binding as means for communication between the mobile devices are used, but this approach requires the ability to host Web services on lightweight devices. Unfortunately Microsoft did not integrate a small footprint Web

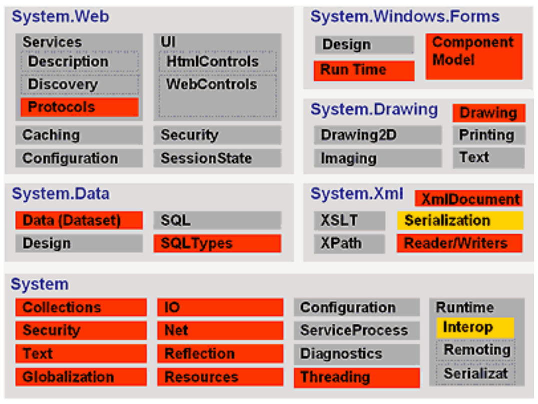


Figure 1: .NET Compact Framework Modules

services provider in the .NET Compact Framework and to the authors knowledge there are no additional packages or commercial products available which enhance it to host Web services.

This significant limitation makes the .NET Compact Framework drop out of the field of competitors of possible platforms for this project.

Java 2 Micro Edition - J2ME

Similar to the .NET Compact Framework the J2ME Platform is an execution environment with a subset of features of the full Java SE platform based on version 1.4.2. Unlike .NET CF there is not only one runtime environment. J2ME is modular and can be adapted to meet the requirements of the developed application in a flexible manner. J2ME is designed to run on embedded devices ranging from high-end Personal Digital Assistants (PDA), high-end Mobile and Smartphones over set-top boxes and printers to low-end Mobile Phones and limited devices. Since the spectrum of such devices is manifold, many developers favor the modular nature of this platform to develop perfect fit applications.

A J2ME execution environment consists of a configuration, a profile and optional packages.

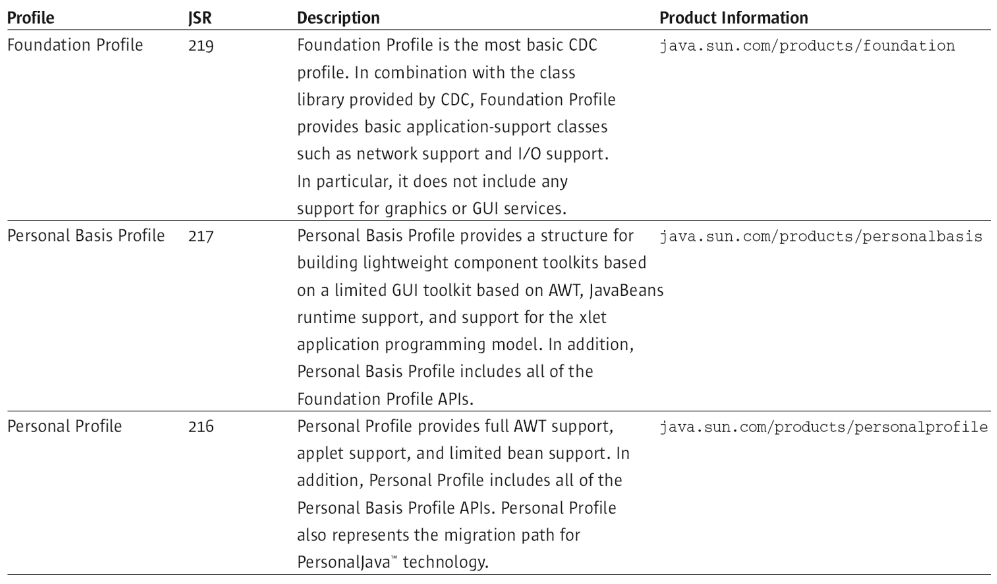
Connected Device Configuration - CDC A Configuration consists of a basic set of APIs and Java virtual machine features. The footprint of an execution environment with the Connected Device Configuration is ranging from around 2 Mb to 9 Mb and thus runs only on embedded devices with more advanced resources such as PDAs and Smartphones. But on the other side, the CDC features the full Java Virtual Machine specification, including full support for class loading and core library features, which enables developers to reuse their knowledge of the Java SE technology and existing libraries and tools for this platform [41].

A CDC based environment has to be started with a Profile, which is an additional set of APIs that support a narrow set of devices [41]. Profiles enable developers to support different categories of devices. Currently, there are three Profiles available for CDC:

- Foundation Profile
- Personal Basis Profile
- Personal Profile

For a short description of the properties of the Profiles [41] see Figure 2.

As we see in Figure 2, the Personal Basis Profile includes all of the features of the Foundation Profile whereas the Personal Profile is set on top of the Foundation and Personal Basis Profile.



Profile	JSR	Description	Product Information
Foundation Profile	219	Foundation Profile is the most basic CDC profile. In combination with the class library provided by CDC, Foundation Profile provides basic application-support classes such as network support and I/O support. In particular, it does not include any support for graphics or GUI services.	`java.sun.com/products/foundation`
Personal Basis Profile	217	Personal Basis Profile provides a structure for building lightweight component toolkits based on a limited GUI toolkit based on AWT, JavaBeans runtime support, and support for the xlet application programming model. In addition, Personal Basis Profile includes all of the Foundation Profile APIs.	`java.sun.com/products/personalbasis`
Personal Profile	216	Personal Profile provides full AWT support, applet support, and limited bean support. In addition, Personal Profile includes all of the Personal Basis Profile APIs. Personal Profile also represents the migration path for PersonalJava™ technology.	`java.sun.com/products/personalprofile`

Figure 2: CDC Profiles

A CDC environment can be further enhanced with optional packages, such as

- RMI (subset of the Java SE RMI, enables remote method invocation and hides network communication and network protocols)
- JDBC (subset of the JDBC 3.0 API, provides means to access tabular data sources)
- AGUI (modified implementation of Swing)
- Security (Security framework based on Java SE)
- Web services (enables access to Web services from Java ME clients)

Connected Limited Device Configuration - CLDC As the name implicates, this Configuration is designed to run on devices with limited capabilities like low-end mobile phones with a typical memory capacity around 600KB of RAM (including the Mobile Device Information Profile, MIDP) and 1,5MB of ROM/Flash. Despite the requirement of a small footprint, the CDC based execution environment must have enough performance to run properly on a device typically featuring a 50 to 200 MHz processor.

In most cases the profile, that is set on top of the CDLC, is the Mobile Information Device Profile (MIDP). Like other Profiles, it is a set of APIs and libraries, that facilitate the handling of the limited resources of the devices the platform is deployed on. Those devices typically have e.g. small display, current entry limited by battery capacity, networking etc. and in version 2.0 even a game API has been included.

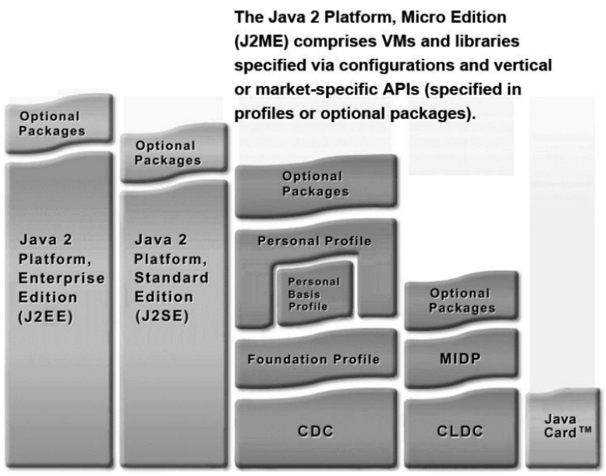

The Java 2 Platform, Micro Edition (J2ME) comprises VMs and libraries specified via configurations and vertical or market-specific APIs (specified in profiles or optional packages).
Optional Packages
Java 2 Platform, Enterprise Edition (J2EE)
Optional Packages
Java 2 Platform, Standard Edition (J2SE)
Optional Packages
Personal Profile
Personal Basis Profile
Foundation Profile
CDC
Optional Packages
MIDP
CLDC
Java Card™

Figure 3: CDC/CDLC Architecture

To summarize, CDC is the choice for current PDAs and Smartphone, whereas CDLC is per definition designed for older and more limited devices like low-end mobile phones, featuring at least a 16-bit CPU and a total of 160 KB (192 KB in CLDC version 1.1) memory available to the Java platform. A limited connection to some kind of network is also required. For a comparison of the most common Java platforms available see Figure 3 [42].

We decided to use at least the CDC with the Personal Profile, because features like Reflection and Class Loading are important to implement a context model.

2.3 Summary

Providing Web services on mobile devices is difficult due to the limited hardware capabilities and restricted software APIs. Mobile devices frameworks today still do not address the role of a lightweight device as a Web service provider, which has thus to be achieved by thirdparty toolkits or frameworks. In contrast to JXTA, SOAP and REST based services allow information providers to be developed independently from information consumers through service negotiation and intercommunications standards. Web services can facilitate the development of distributed applications in a heterogeneous environment, which is the common case in mobile computing.

3 The Role of SOA and Web Services in Mobile Computing

3.1 Technology Overview

SOA Paradigm

A service in terms of middleware is a procedure, method or object with a stable and published interface that can be invoked by clients [39]. Here it is important to say, that the invocation is done by a program running on the client side. A Web service can be described similarly but as the name implicates there is an additional requirement, namely the possibility to invoke the service over the Web. Web services are often developed and managed independently since services can be provided by different companies. Thus, applications following the SOA paradigm (Figure 4 [39]), are constructed of loosley coupled Web services in most cases.

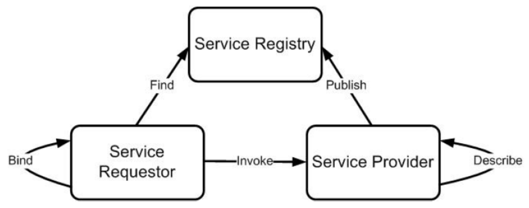


Figure 4: SOA Paradigm

Service registration

To make Web services known to possible consumers, it requires a mechanism to publish the description of Web services. This is achieved by a service registry (e.g. UDDI - Universal Description, Discovery and Integration). Services, that are present in the service registry can be looked up regarding certain criteria and the registry returns the endpoint of a Web service to the client.

Service binding

Once the client has retrieved a proper endpoint from the service registry, it has to negotiate the communication method with the Web service. WSDL (Web Service Description Language) is an appropriate means for this purpose. Based on the information of a WSDL document of a Web service the client is able to interact with the service. To sum it up, the service binding is the negotiation of the message format the service understands and the protocol atop which those messages are transported.

Service invocation

When the service binding is complete, the last step of the client is the invocation of the service and the retrieval of the results. The WSIF (Web Services Invocation Framework [26]) facilitates the invocation of Web services for Java-based applications dramatically. It enables the developer to interact with the service through a simple Java API. The only requirement is that the service is described with WSDL. The invocation can be done stubless and completely dynamic at runtime.

Web Services

Web services are part of the Service Oriented Architecture paradigm and consist of a set of XML based standards [36], to interconnect loosley coupled applications mainly over the WWW. There are various definitions of the term "Web services", ranging from very generic to very specific [39]. An example for the first case would be any application that is identified in the web through an URL (Uniform Resource Locator) and accessible to other applications. Such an application could be a simple script or a more complex service, where APIs are provided (e.g. Amazon Web services).

The W3C defines the term Web service as follows [25]:

> A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via internet-based protocols.

Web services are kind of a buzzword nowadays and have evolved along the expansion of the Internet infrastructure. Because of the ubiquity of the web and broadband everywhere, Web services have become a broad field of research in computer science.

Message Encoding

In most cases Web services make use of XML as a message encoding format, because it is flexible, human readable and a standard for platform independent data interchange. XML stands for eXtensible Markup Language. It is the direct successor of the SGML and became a W3C recommendation on the 10. February 1998. XML is a text-based markup language mainly designed to describe data of various kinds as opposed to HTML, which has been designed to display documents in a web browser. XML is not a programming language, it simply structures and stores data, which can be processed by machines. Unlike in HTML, the tags of XML are not fixed and can be defined freely to fit the user's requirements. As we can see in the following example, the tags in most XML documents are designed to be human readable too. Listing 1 shows an example of an XML document.

Listing 1: Example XML Document

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<note>
  <to>Mark</to>
  <from>Mary</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

WSDL

WSDL (Web Service Description Language) is a language to describe Web services in terms of their operations, messages, datatypes and communication protocols [10]. WSDL files are defined in XML.

Types This element defines the datatypes, which are used by the Web service.

Example:

Listing 2: WSDL Element

```
<xs:element name="addRecord">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="artist" type="xs:string"/>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="titleID" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 2 defines a complex element named `addRecord`, which consists of three subelements that hold the information.

Message This section of the WSDL file defines the messages which are used by the PortType definition to describe the data format.

Example:

Listing 3: WSDL Message

```
<message name="addRecordRequest">
  <part name="parameters" element="wsdl:addRecord"/>
</message>
<message name="addRecordResponse">
```

```
  <part name="parameters" element="wsdl:addRecordResponse"/
     >
</message>
```

In the message section of the WSDL file the messages passed between the Web service requester and provider are defined. We can see in Listing 3 that the above defined complex element `addRecord` is used as a parameter for the message `addRecordRequest`. The message `addRecordResponse` also refers to an element as parameter that has to be defined in the elements section.

PortType The PortType describes the Web services operations and the messages, which are used to exchange data.

Example:

Listing 4: WSDL PortType

```
<portType name="MusicStoreServicePortType">
  <operation name="addRecord">
    <input message="wsdl:addRecordRequest"/>
    <output message="wsdl:addRecordResponse"/>
    <fault name="duplicateEntry" message="
       wsdl:duplicateEntryFault"/>*
  </operation>
  <operation name="removeRecord">
    <input message="wsdl:removeRecordRequest"/>
    <output message="wsdl:removeRecordResponse"/>
    <fault name="recordNotFound" message="
       wsdl:recordNotFoundFault"/>*
   </operation>
</portType>
```

This `PortType` defines two operations `addRecord` and `removeRecord`. Each operation has an input and an output whose format is defined by messages. In this case it is the above defined `addRecordRequest` and `addRecordResponse` message. In case of a failure the `duplicateEntryFault` message is returned to the Web service requester.

Binding The Binding section defines the protocol and message format for each port.

Example:

Listing 5: WSDL Binding

```
<binding name="MusicStoreServiceBinding" type="
   wsdl:MusicStoreServicePortType">
  <soap:binding style="document" transport="http://schemas.
     xmlsoap.org/soap/http"/>
  <operation name="addRecord">
    <soap:operation soapAction="http://musicstore.com/
       addRecord"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
    <fault name="duplicateEntry">
      <soap:fault name="duplicateEntry" use="literal"/>
    </fault>
  </operation>
</binding>
```

This binding defines that the Web service is using HTTP as a transport protocol and SOAP as the message format.

3.2 Web Services Toolkits

gSOAP

gSOAP is a SOAP toolkit implemented in C++ and intended to be platform independent [51]. It is an approach for a SOAP-to-C++ language binding for being able to realize C++ application as a SOAP based Web service. gSOAP is fast, reliable and secure and predestinated for being deployed on mobile devices. It features a WSDL parser and generator, and also a stub/skeleton compiler. Due to high interoperability with other SOAP toolkits gSOAP is the best solution to realize SOAP based Web services on devices with limited capabilities (implemented in C++). As gSOAP is written for C++ it was unfortunately not usable for the proof-of-concept implementation presented in this work. We use OSGi as a deployment container for the framework, which is set atop the Java platform.

SOAP and J2ME

To deploy and manage services on a mobile Java platform it is recommended to use one of the frameworks available (e.g. OSGi, Jade). The act as a container for applications and provide services (e.g. OSGi provides a HTTP server) which is shared among all applications in the container. In this project, the OSGi middleware is used as a container (See Section 3.3 for a detailed description of OSGi).

The focus on Web services on mobile devices as communication means between the containers required an investigation and evaluation of

Web service toolkits for J2ME. Unfortunately there are not too many competitors in the field of Web service providers for J2ME and all of them suffer from different shortcomings. Schall et al have done a comparison of Web Services on Embedded Devices [36]. Based on this work this section deals with Web service toolkits for J2ME and shows their advantages and disadvantages. Experiences made while experimenting with the toolkits are added to the description of each technology.

Apache Axis

There are two Web service bundles available for the Knopflerfish OSGi framework. One of them is the Apache Axis bundle, which was originally written for J2SE. It is port of the standard Apache Axis server which requires a servlet version 2.3 environment, whereas the Knopflerfish framework only provides a servlet version 2.1 environment. During porting the Apache Axis server to OSGi, no changes have been made to the Apache Axis classes, so all code specific to the OSGi port is made as pure “extensions” [22].

Due to te fact that it was not intended to run on a limited environment, it has a very large footprint, but also the greatest functionality. It allows to expose any service registered in the OSGi framework as a SOAP service. The only constraint is that the service must not expose any datatypes not supported by SOAP. Apache Axis also provides the possibility to generate WSDLs from its Web services. It can be run on the IBM J9 JVM, but only by adding additional boot classes borrowed from the J2SE 1.4.2 JRE. The missing classes must be included in the bootclasspath of the JVM and explicit access to those classes must be allowed in the initialization file of the OSGi framework.

Though Apache Axis can be deployed on a lightweight device on the IBM J9 JVM, it is definitly not the best option, because the footprint is simply too large. The Apache Axis 1.4 OSGi bundle with all required libraries consumes approximately 2 Megabytes of system memory.

kSOAP-osgi

The second bundle available for Knopflerfish OSGi is the ksoap-osgi bundle. The bundle is based on kSOAP2, which is a SOAP library for constrained Java environments, such as applets or J2ME applications (CDC/CDLC/MIDP) [18]. The bundle has been developed to be able to remotely control Knopflerfish frameworks running on headless or embedded devices.

Besides this functionality, it is also possible to register SOAP servlets in the HTTP server of Knopflerfish. Those servlets (or better services) have to extend the SoapServlet of the kSOAP2 package. Primitive parameters and return values can be transferred directly, whereas complex datatypes have to be constructed manually, complex return values must be parsed manually and envelops for SOAP calls have to be generated manually. The functionality of this bundle is very basic, some useful features like automatic envelope generation and response message parsing are missing.

jSOAP

jSOAP is a implementation of SOAP 1.1 for Java, which allows providing and consuming Web services. Due to its small footprint it is suitable for deployment on devices with limited capabilities. It requires a servlet environment, which is provided by the Knopflerfish framework, but the documentation lacks the information, if version 2.1 is sufficient. The servlet required for hosting a Web service is not generated automatically, it has to be written manually. Furthermore, the services exposed have to be configured seperately by an XML configuration file, which is parsed on startup of the servlet. This file contains the service and parameter names, the parameter types and count and the expected type of the return value to ensure a proper mapping of the service to the Java interface. jSOAP also features a WSDL generator, which is helpful to expose an abstract description of the service. On the whole, jSOAP appears to be a sophisticated toolkit with lots of features, but the concept is not as convenient for the developer due issues like configuration efforts, manual request generation and transmission etc. as the approach mentioned below.

AMIGO

Amigo (full project name is “Ambient Intelligence for the networked home environment”) is a middleware based on OSGi, which was developed by a joint venture of fifteen european companies and research establishments in mobile and home networking, software development, consumer electronics and domestic appliances. The goal of Amigo is to make visible the full potential of home networking and to overcome issues like complex installation procedures and interoperability problems between heterogenous components of a networked home environment [1] and to lead this new technology to a broader acceptance. The result is a middleware of open, standardized and interoperable services based on OSGi.

The services are packages as OSGi bundles which makes it very easy to select and deploy specific services. The most interesting part of the middleware for this work was the Amigo Service Exporter, which provides the possibility to export Java objects as SOAP based Web services. There are two bindings available for the service exporter, a kSOAP binding and a Apache Axis binding. Due to the above mentioned problems of Apache Axis running with the IBM J9 JVM, the kSOAP binding was used for this project.

The service exporter makes extensive use of the Reflection API while registering any object as a Web service. This is very convenient for the developer, because no additional service descriptions except the class definition is needed. To the authors knowledge, there is no other toolkit available enabling the developer to provide Web services in such a simple way.

The Amigo workgroup provides a tutorial [2] how to use the middleware, which seems to be sufficient at first sight, but with ongoing intensive engagement, missing details become obvious. Enabling the dynamic stub generation mechanism was not possible, because this feature is unfortunately not documented in the tutorial. Also the handling of datatypes, except the simple string example in the tutorial, is not mentioned. Lack of documentation is the biggest disadvantage of the Amigo middleware.

3.3 Service Deployment

OSGi

Making Web services available on a mobile device requires a suitable platform. One approach for this task is OSGi. OSGi is a specification proposed by a consortium of various companies like Ericsson, IBM, Nokia etc.

The OSGi Alliance introduces to OSGi as follows [30]:

"The OSGi^{TM}specifications define a standardized, component oriented, Computing environment for networked services that is the foundation of an enhanced service oriented architecture."

An OSGi framework is a lightweight Java-based platform, which provides many standard services like Security, Logging, Configuration Management, HTTP service etc. and allows the convenient development of service oriented software applications for embedded devices. Those applications are encapsulated in so called "bundles", which can be installed, started, stopped and removed on the fly, without interrupting the operation of the device or the rest of the framework. One big advantage resulting from this is the fact that only one Java Virtual Machine is needed for multiple Java based applications. Security and life cycle management is done by the framework implementation (Figure 5 [30]).

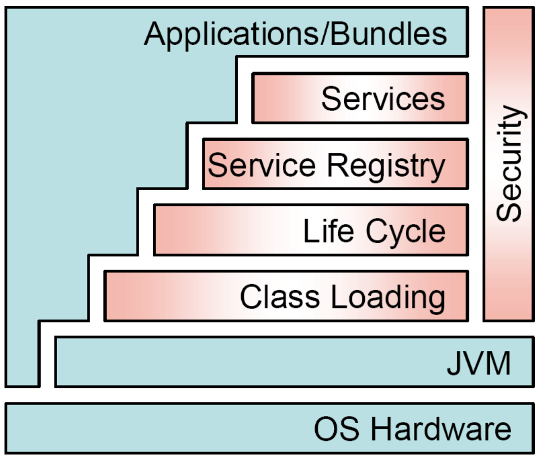


Figure 5: OSGI Architecture

Bundles can make use of the services provided by the framework itself or by other installed bundles and are able to provide their own services. OSGi features a service discovery mechanism, which allows bundles to listen for service registration events using LDAP filters. Once such an event is triggered, the listening bundle is notified, and code based on the type of the event can be executed.

Due to the bundle structure and dynamic life cycle management, OSGi is the perfect partner when it comes to modularity. Libraries and services can be easily encapsulated into multiple separat bundles, which helps to maintain the outline in a complex software project.

Popular implementations of the OSGi specifications are

- Oscar [20]
- Equinox [4]

- Concierge [3]
- Knopflerfish [16]

All those Implementations require at least a JVM with CDC and the Personal Profile. There is also a commercial implementation of OSGi for the CLDC made available by the german company ProSyst called "mBedded Server CLDC Edition" [21].

The OSGi-Bundle

An OSGi-Bundle is a JAR (Java Archive) containing the Java classes and a MANIFEST file. The bundle can be added to the OSGi framework either during bootstrapping or later at runtime using the console or a user interface. When a bundle is being installed, the MANIFEST is parsed. This file contains metadata about the bundle e.g.

- Name, Version
- Bundle classpath, pointing to included resources in the JAR (e.g. libraries)
- Package imports
- Package exports
- Dynamic imports
- Link to the bundle activator

The use of the name and the version of the bundle is obvious, the name is used for identification and the version is used for versioning purposes. More interesting are the next 3 points on the list.

The bundle classpath points to resources included in the bundle, which are needed to execute the contained application. Such resources can be JARs or directories.

In the package imports section, the packages that are required to run the bundle are listed. Note that these packages are not included in the bundle itself, they must be provided by the framework or by other bundles. The OSGi framework tries to resolve all these dependencies at startup. If they cannot be resolved, the installation of the bundle fails.

A bundle can export any packages that are included in its classpath. These packages are made available to other bundles in the framework. It is a mechanism, that is very useful for purposes like sharing a library with other bundles. In this case, the library only has to be included in one bundle but can be imported by others.

Dynamic imports are packages that are not verified by the framework whilst installing the bundle. This mechanism has to be used with care, because the programmer has to ensure, that those dynamic packages are available when the bundle is started, otherwhise the framework terminates due to a NoClassDefFoundException.

The bundle activator is a class that implements the BundleActivator interface of the framework implementation. It is called when a bundle is started (not on install) and responsible for the proper initialisation / startup and proper shutdown respectively. A bundle activator is not mandatory. Bundles without activator can be seen as resource bundles, but they only make sense, if they include resources or packages that are shared with other bundles. Otherwhise it would only increase the footprint of the framework deployment.

OSGi Services

One of the benefits of the service oriented architecture of OSGi is that bundles can not only import packages provided by other bundles or the framework itself, they can search for services using standard LDAP filters and make use of them as long as they are available. Yes, if you see a problem here, you are right. This can be tricky sometimes. Services always have to be tracked and checked for availability, because the bundle providing the service can be removed at runtime. This fact poses the requirement of a robust and fault tolerant programming style to the application developer. The status of services can be tracked using the events mechanism of the OSGi framework. Bundles can listen to events concerning any active service using the LDAP filters again. On the basis of the event types decisions can be made (e.g. stop the bundle etc.)

Besides the consumption of services, it is of course also possible to provide services. A OSGi service is basically any Java object instance, that is registered with the framework using a name and properties (for LDAP filtering and service selection) but an instance of an implementation of a well-known interface would be tidy.

But what if an object must not be used by multiple bundles? OSGi also has a solution for that - the service factory. This mechanism creates a new service instance for every bundle requesting a service.

Managing service dependencies can be very complex and laborious sometimes, depending on the number of service that are used in a bundle. I will get back to this point later.

For more information on OSGi services and their use, please refer to the documentation of the particular OSGi implementation you are using. Knopflerfish provides an excellent tutorial on OSGi services [16].

3.4 Service Discovery

In a service oriented architecture the process of finding a service in the network meeting certain requirements is called Service Discovery. Basically there are at least two actors involved in the discovery process. The Service Consumer wants to query a certain service and is starting the search on the network. This can be either done by a broadcast based protocol, where the Service Provider is answering if the criteria are matching or by a (well known) Service Registry, where Service Provider register their services and endpoints. This Service Registry can be used by Service Consumers to look up the service needed.

In a mobile environment, several challenges arise. Because of the mobile nature of portable devices, it is unpredictable when those devices go on- or offline. Thus, if a service registry is used for the discovery, there has to be a mechanism to manage the orphaned service entries. Imagine the case that a mobile device has registered a service in the registry. Imagine also, that this device moves unpredictably out of network coverage after the registration. Now the Service Registry holds an entry that is pointing to a device that is not available anymore. Another problem is, that in a mobile environment is is likely that a device has no access to a service registry. In this case it cannot make use of services that devices in proximity are providing.

Thus, using a broadcast based approach seems to be the better solution for ad-hoc networks. Service providers are listening on the network and responding to requests if the criteria match. The service consumer is not able to invoke the service, which is most likely available. In this section three Service Discovery Protocols will be presented.

Service Discovery Protocols

Ubiquitous computing and service oriented architectures in mobile computing require means to discover services provided by mobile participants in ad-hoc or infrastructure networks. Many service discovery protocols have evolved like

- INS
- Ninja SDS
- IBM DEAPspace
- Jini
- UPnP
- Rendezvous

- Salutation
- SLP
- Bluetooth SDP

which support the advertising, discovery and usage of services. As I needed such a mechanism for my project, I investigated three service discovery protocols and examined them for suitability.

Service Location Protocol - SLP

The Service Location Protocol standard is being developed by the Service Location Protocol working group (SRVLOC) which has been active in the Internet Engineering Task Force (IETF) for several years [40]. The first version of the SLP was published as a Proposed Standard RFC in 1997. Two years later, in June 1999, the group announced version 2 of SLP, which replaced version 1 and is still the currently used standard.

The SLP enables applications on networked devices to search for available services on the network. The protocol allows not only the retrieval of the service name, the application using it is able to obtain location, address, domain name or other configuration details. SLP does not restrict the discovery to types of services. Hence it is possible to search for services using certain characteristics. If a service has been discovered, SLP allows a detailed lookup of its attributes.

Components The “actors” using the Service Location Protocol are called agents:

- User Agent (searches for a specific service on the network)
- Service Agent (advertises a service on the network)
- Directory Agent (collects information about available services and responds to discovery requests)

Discovery For service discovery, SLP provides several operational modes. First of all the most intuitive case is a client searching for a certain kind of service. The client (User Agent) sends a multicast querying for the service he wants to use, and waits for an answer from online service providers (Service Agent). If a Service Agent can fulfill the User Agent’s request it sends a unicast telling the client that it provides the service. From now on the communication between the two parties is done via unicast.

The second case is a kind of “passive” service discovery. When a Service Agent joins the network, it announces (advertises) its services via multicast to other peers. User Agents listening for new service announcements can judge if they need the new service or not. In case the new service is exactly that the peer has been waiting for, he responds to the new Service Agent via unicast and invokes the service. As in the first case, from now on communication is done via unicast.

Discovering a Directory Agent can be achieved in the same manner mentioned above. A Directory Agent can be described as a special

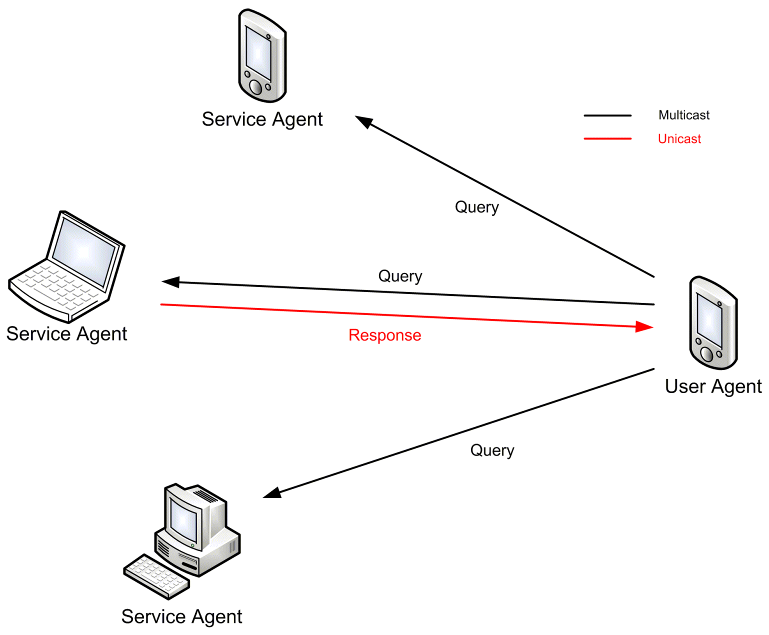


Figure 6: Active Discovery using SLP

kind of Service Agent, which provides a service for easier discovery of other Service Agents. It helps to reduce the network load, because User Agents do not longer have to multicast their service discovery requests. Another method of making Service and User Agents aware of a Directory Agent is using the Dynamic Host Configuration Protocol. Peers can be updated with the location of a Directory Agent whilst obtaining an IP-address. If a Directory Agent is present and known to the peers it acts like a dispatcher for services. Discovery requests and advertisements are not longer multicasted over the network, they are unicasted to the Directory Agent, which stores new services and their locations or tells User Agents, where they find services with requested types and properties. Figure 8 shows the two typical methods of discovering a Directory Agent [40].

Service Agents using SLP are requested to register their services with

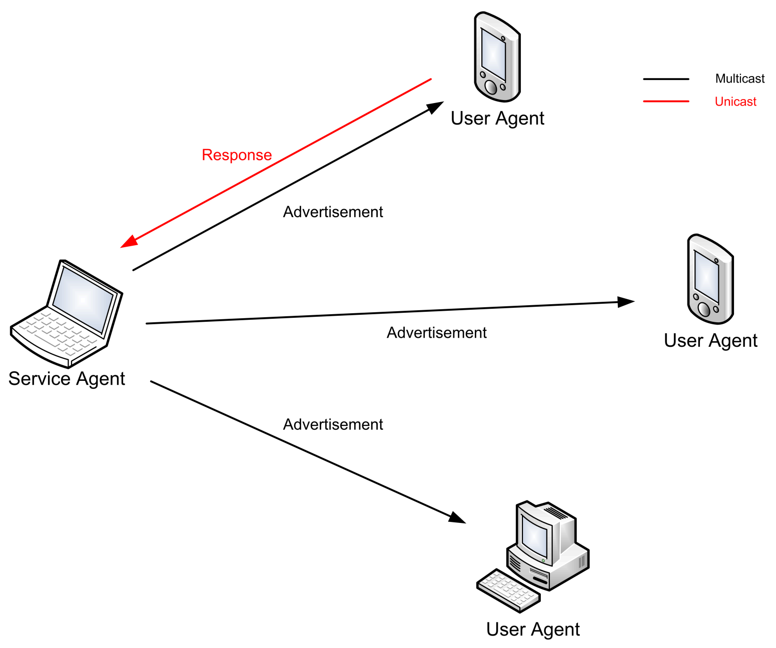


Figure 7: Passive Discovery using SLP

a Directory Agent, if one is present. It is obvious, that in a frequently changing mobile computing environment, where peers are loosely coupled technical difficulties like network problems can arise. The question is now, how to handle for example an interrupted connection whereby a Service Agent becomes unavailable whilst its services are still registered with the Directory Agent. SLP addresses this issue with timeouts. Services registered with a Directory Agent have a timeout which must not be set higher than 18 hours. Service Agents thus have to renew their service registration periodically to ensure proper lookup. Service Agents leaving the network should send a Service Deregister message of their services to the Directory Agent. If a Service Agent fails or leaves without deregistering, the timeout takes care of orphaned services.

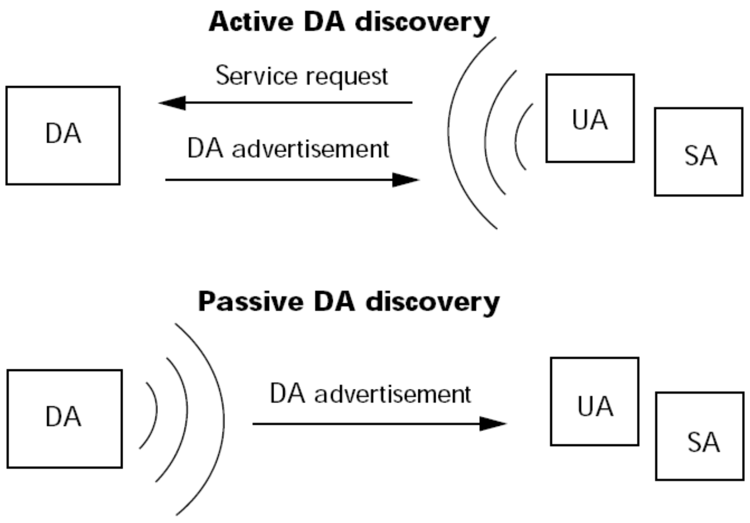


Figure 8: Typical DA Discovery

Multicast Convergence Algorithm The Multicast Convergence Algorithm helps to improve the scalability of the Service Location Protocol. Imagine a User Agent requesting a service. All Service Agents providing a matching service are responding. If the same request is issued again after a certain period of time, the client will most likely get a large amount of answers from the same Service Agents, which causes unnecessary traffic on the network. The designers of SLP solved this problem by letting the User Agent transmit a previous responder list. When a Service Agent gets a request, he first checks if he is on the list. If so, he does not respond to the client. This method helps to keep the amount of unneeded messages very low.

SLP is a very lightweight discovery protocol, which is mainly string based. Services can be browsed by types and attributes, which is sufficient for our project. A typical Service URL is for example:

```
service:type:protocol://server-address:port/endpoint
```

This is what a Service Agent returns to a User Agent in case of a match. As we can see here and as it is hinted in the name of the protocol, it only allows the retrieval of service addresses. If a client wants to access a service, the parameters needed have to be negotiated with the Service Agent in advance.

WS-Discovery

WS-Discovery is another specification of a service discovery protocol, which is supported by famous companies like Microsoft, BEA, Canon, Intel and webMethods. As SLP, the protocol allows the retrieval of service locations in high dynamic networks, with frequently changing peers. To achieve this, WS-Discovery uses the same means as SLP, as it takes advantage of the multicast mechanism to reach a broad group of peers on the network.

Discovery WS-Discovery provides two types of services discovery requests. A service can be looked up by type or by name. When a client joins a network and needs a certain service which is only constrained by a type, he sends a probe message via multicast to the peers on the network. Target services, which match the probe message, respond directly to the client. But services can also be resolved by name. The procedure is the same as above, only the request sent by the client differs slightly. In this case it is a resolution request message, which is answered by the target service matching the name (Figure 9 [43]).

Similar to SLP WS-Discovery addresses scalability with two nearly identical mechanisms. The first can be compared with the Advertise-

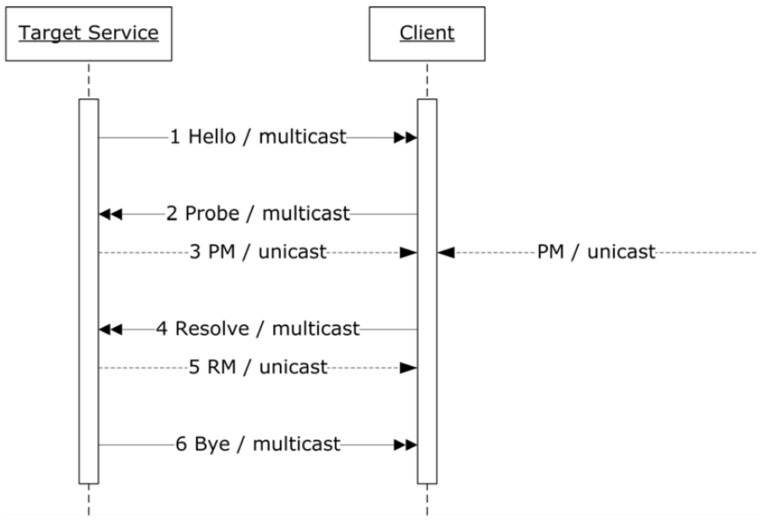


Figure 9: Service Discovery using the WS-Discovery protocol

ment in SLP. A client providing a certain target service joining the network sends an announcement message. This mechanism ensures that clients, which probably need a service, are always kept up to date by listening to those announcement messages. Network traffic is reduced because frequent probing to check the network for service changes is not necessary anymore.

The second mechanism is the Discovery Proxy. The WS Discovery Proxy is a kind of service, which keeps a list of available target services on the network. By listening to probe or resolution messages it detects clients, which do not know about its existence. If so, it informs the client about the presence of a Discovery Proxy, causing the client to switch to the Discovery Proxy specific protocol. Service requests are from now on not sent by multicast, the lookup is done directly at the discovery proxy, which lowers the number of multicasts significantly (Figure 10 [43]).

In case the Discovery Proxy fails or leaves the network, the clients

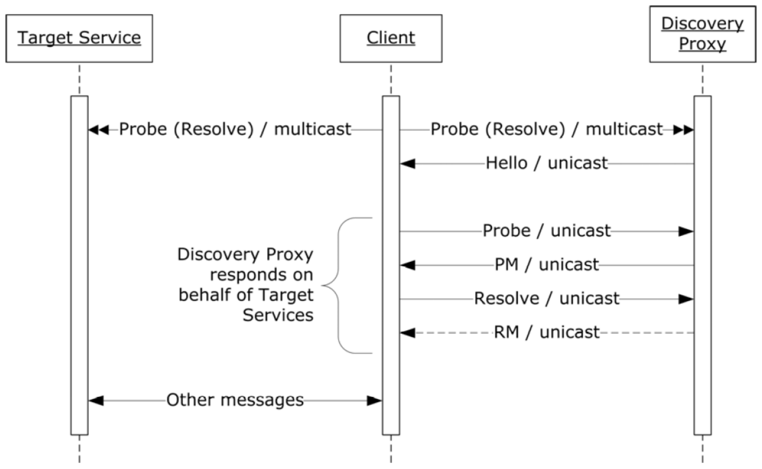


Figure 10: Service Discovery with Discovery Proxy present

switch back to multicast discovery. The default timeout value in the specification is 5 seconds.

In contrast to SLP, WS-Discovery does not use a proprietary message format. The clients communicate using the Simple Object Access Protocol (SOAP) which is based on XML. Exact examination and description of the messages is beyond the scope of this work. For further information the reader is advised to take a look at the WS-Discovery specification [43].

Universal Plug and Play - UPnP

The UPnP Forum introduces to Universal Plug and Play as follows [47]:

“UPnP technology defines an architecture for pervasive peer-to-peer network connectivity of intelligent appliances, wireless devices, and PCs of all form factors. It is designed to bring easy-to-use, flexible, standards-based connectivity to ad-hoc or unmanaged networks, whether in the home, in a small business, public spaces, or attached to the Internet. UPnP technology provides a distributed, open networking architecture that leverages TCP/IP and the Web technologies to enable seamless proximity networking in addition to control and data transfer among networked devices."

As we can see here, the presented technology is intentionally very similar to the two discovery protocols discussed above. It also enables peers to announce their services and system properties and offers the possibility to search for certain services on the network. But UPnP is more powerful than WS-Discovery and SLP as it features eventing and presentation of the devices.

The actors in UPnP are called controlled devices (or just simply devices) and control points. A "device" can be compared with the Target Service in WS-Discovery and with the Service Agent in SLP. Control points represent the client side and are very similar to clients in WS-Discovery and User Agents in SLP.

The major elements of UPnP are

- Step 0: Adressing (Clients get a network address)
- Step 1: Discovery (Clients find interesting devices)
- Step 2: Description (Clients learn about the capabilities of found devices)
- Step 3: Control (Clients access found devices and send commands)
- Step 4: Eventing (Clients listen to the state of found devices)

- Step 5: Presentation (Devices expose a user interface)

Each element is depending on the elements listed above. If a control point wants to control a device, it is necessary that it has a network address to be able to communicate with the devices. It is also necessary that it has discovered the device it wants to control. Additionally it has to be able to lookup the way it should access the service via description. So step 3 (Control) requires at least step 0-2 to work properly.

Addressing To be able to communicate with each other, hosts need to have an IP address assigned. This can be achieved in two ways. When a client joins a network, he first tries to find a DHCP server. If one is found, a valid IP address is obtained, and the client is ready for starting discovery. If no DHCP server is found, the client auto-configures using Auto-IP, which is simply a randomly selected IP address in the 169.254/16 range. To avoid collisions, the client has to test, if the chosen IP address exists on the network. In case another peer is already using the selected address, the client tries another IP address in this range using an implementation dependent algorithm until he gets a free one. If Auto-IP is used for addressing, the client must periodically check the availability of a DHCP server to obtain a “real” IP address, to be able to use the network infrastructure to the full extend.

Once the client has obtained the IP address, it is able to advertise its own services or to search for services provided by other peers in a conceptually very similar manner as SLP and WS-Discovery. Basically there are two methods to discovery services. The client can listen for advertisements on a standard multicast address and judge which services announced are of interest for it. This can be seen as a sort of passive discovery as opposed to “active” discovery, where a client sends a message with search parameters to discover a required service. Detailed description of the messaging and discovery algorithms shall not be mentioned here. For further information please check the UPnP specification [47].

Description After having discovered a device, the description of the device and its services becomes important. Via description the client gets vendor-specific information about the device itself, definitions of the embedded devices, the URLs for control, eventing and presentation and a list of all exposed services. Description is vital to tell the client how to interact with the discovered devices.

Control When a client has discovered a device and queried the description to get interaction information, it is ready to invoke the desired service. This step is called control in UPnP. Control commands are sent as SOAP messages using HTTP. After having issued a command, the client receives either a result in case of success or a failure message, if the operation couldn’t be completed successfully from the controlled device.

Eventing Eventing is a mechanism to monitor the state of controlled devices. After Discovery and Description (Step 1 and 2) the client has retrieved the endpoint URLs for Eventing and is now able to subscribe to events produced by the monitored device. After receiving the subscription the controlled device sends a message with the current state variables in the XML format and a time value how long the subscription will be active to the client. In this time period the subscriber will receive continuous updates of the state variables. After the time-to-live the subscription has to be renewed by the client, else the controlled device stops sending updates.

Presentation After the UPnP device has transmitted the client the Presentation endpoint URL, the control point can retrieve the presentation web page into a suitable browser. This page is completely vendor specific, it can be implemented to either monitor the status of the UPnP device or control the device to a certain extend. The only constraint for the presentation page is the format, which has to be HTML version 3.0 or later. Design and implementation issues are left to the vendor. An example for such a presentation page would be the web interface of a consumer router, where settings can be configured using a web browser of choice.

Summary

SLP is a binary protocol, which uses key-value pairs as description mechanism. WS-Discovery and UPnP are based on XML and have all the advantages and drawbacks XML based protocols have compared to binary protocols. Due to the verbose nature of XML, XML-based protocols are much easier to debug, because the messages sent are human readable in contrast to protocols, which rely on binary messages. The great extensibility and flexibility of XML is dearly bought with a large overhead, that is caused by the XML format. For the proof of concept implementation presented in this work, SLP was used. The powerful description mechanisms of WS-Discovery and UPnP are not necessary, because every participating device in the network intending to query a ContextStore has to carry the API bundle, in which all necessary information to access the service is included. The only thing that must be known is an appropriate endpoint.

4 Case Study - Collaboration in Mobile and Ad-hoc Teams

4.1 Context-Aware Mobile Computing

Context

To be able to understand what context is and how it can be utilized for collaboration environments, we have to take a look at existing work in the field of context aware and ubiquitous computing. According to Dey et al. [28] the term “context-aware” was first introduced by Schilit and Theimer [31] who state that context can be referred to as location, identities of nearby people and objects and consequently changes to those objects. Many other definitions have been made by other authors, but they only enumerate what belongs to context from their point of view, which makes the determination of context very difficult. What’s included in context information or not changes constantly with the situation a user resides in. Therefore a simple enumeration is not very applicable in my opinion. Synonyms or too specific definitions are also not very helpful because they are extremely hard to apply practice [28]. Schilit et al. [32] lead Dey et al. [28] to their definition context as they see context as a question of “where you are, who you are with, and what resources are nearby”. Furthermore context is “the constantly changing execution environment”. They include the following aspects of the environment:

- Computing environment e.g. available processors, devices accessible for user input and display, network capacity connectivity and costs of computing

- User environment e.g. location, collection of nearby people, and social situation
- Physical environment e.g. lightning and noise level

Dey et al. [28] define context as follows:

> "Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

This definition makes it very easy for the application developer to determine whether a piece of information can be considered as context or not. In addition they state, that researchers exploring context have concentrated their efforts on implicit information pertaining because they see still more potential of this approach to enhance the human computer interaction. Dey et al distance themselves from this point of view as they consider also explicit information as a part of context.

Context modelling

To be able to make use of context information in any application, it is inevitable to deliberate a well designed context model. According to [54] early approaches to context modelling have mainly been designed for the usage with one application. As one can imagine such proprietary models are not very desirable when it comes to context sharing between multiple applications or application classes, which is the reason why current research is concentrating on generic models to enable context aware applications to interoperate with each other. A more detailed discussion of context modelling would be out of the scope of this work, as the focus lies on architectural concerns of the whole framework not on the modelling of context. For further information the reader is advised to take a look at the context modelling survey of Strang and Linnhoff-Popien [54].

Related Projects

Many efforts have been made to take advantage of context information in collaborative environments because context aware applications are smarter than conventional software products and are rather able to support a user in his everyday work.

A project, which has similar goals as the framework, I am proposing in my work, that means technical and functional flexibility, is UbiCollab [46]. It aims at satisfying the needs of various different mobile workers strongly depending on the type of cooperation between them. Another focus lies on the compatability of the proposed collaboration environment with new hardware available on the market. Those two aspects played an important role while constituting the requirements of my framework, such as modularity, platform independency etc.

The architecture of UbiCollab is somewhat different. The authors rely on a client server architecture and have implemented an API for extending the service platform with plugins, whereas I am favouring the service oriented architecture. For communication XML-RPC is utilized, having the disadvantage that there are no XML-RPC servers (known to me) running on lightweight devices. This implicates that the device running the UbiCollab service platform must be a rather heavyweight machine with enough resources to bear the XML-RPC server. This is the main difference between the two works, because I need a way to provide lightweight Web services on lightweight devices.

Hydrogen [52] is another approach to make applications on mobile devices context aware, which addresses issues like the lightweightness of mobile devices (lack of system resources, computing power, memory etc.), extensibility, robustness, meta-information and context-sharing. The work was done a few years ago, so the hardware has changed a bit until now, but the workgroup around Hofer et al. nearly used the same device configuration (J9, WinCE, Pocket PC). The framework is split into three layers (Application, Management, Adaptor) and each participating device has its own ContextServer. When a device detects an other device in proximity the two servers exchange their context information. This transaction is called context-sharing. The architecture differs here drastically, because in my framework only few (but at least one) devices need to have a context server (ContextStore) installed. The acquisition and communication process between client and server is completely hidden. Clients can use the remote context server as if it was local. Communication in Hydrogen is done with a proprietary XML protocol.

4.2 Context Model

The context model is designed in UML which can be mapped to various resource description languages like RDF or OWL. These languages are defacto standard for context modelling as they store entities and their relations and are thus capable of reasoning. On the other hand UML can be mapped to object oriented languages like Java, which is used for this project. Following the object oriented paradigm we can make use of the benefits of object oriented technology like encapsulation, reusability, inheritance etc. The main reason to decide in favor of the object oriented paradigm is that toolkits for RDF or OWL are rather heavyweight and reasoning requires a lot of resources. Due to the limited capabilities of mobile devices this approach did not seem to be appropriate.

To store the context information there are also several posibilites. The use of XML is an option, as it has several advantages. Sharing of information is easy as it can be done by simply exchanging the whole XML document. The structure of the document can be described with XML schema and information can be retrieved using technologies like XPath. The drawback of XML is again the large overhead and that it becomes computationally intensive as larger as the file gets, which is not appropriate for the target devices. Using SQL is another option. There are several implementation for SQL databases available for Java which are lightweight and fast. SQL also provides select and filtering mechanisms for data retrieval. Databases also manage concurrent access automatically which ensures consistend data. In this project the hSQLDB [7] for data persistency is used.

Use Case

To deliberate a context model it is always helpful to have a use case where most of the entities and their relations can be extracted. The assumption is that all actors and their skill profiles of the following use case are known to the system. The management of a company has decided to give Steve the supervision of TASK X. To accomplish TASK X a number of activities have to be completed. Steve generates a list and inserts the activities into the system. For ACTIVITY 1 Steve has to meet with COWORKER 1 to discuss several details of ACTIVITY 1. He looks up the current location of COWORKER 1 and realizes that he is only a few miles away from Steve's current position and decides to arrange a personal meeting. TASK X is very time critical, so Steve has to find out the fastest way of contacting COWORKER 1. He queries his current status and sees that he is currently online. Another query delivers Steve the communication capabilities of COWORKER 1 and his preferred device. It turns out that COWORKER 1 wants to be contacted by GoogleTalk. Steve writes an invitation and due to the fact that COWORKER 1 is online he receives an acknowledgement immediately. He meets with COWORKER 1 at LOCATION X and starts the discussion of ACTIVITY 1. During the meeting the system supports the retrieval of related artifacts stored anywhere on participating devices. At a certain point their opinions diverge and they decide to request expert knowledge from a specialist of their company. The system provides a mechanism to search for users having certain skills or even skill profiles, so it's rather simple for them to find an expert having the knowledge they need to successfully complete the discussion about

ACTIVITY 1. They pick out the most suitable candidate for this task (=EXPERT 1) and initiate the contact as described above. The lookup of EXPERT 1's location tells Steve and COWORKER 1 that he is not able to join the meeting, but they find out that EXPERT 1 has the possibility to join a video conference. The system confirms that the currently used device of EXPERT 1 supports video conferences and the network has enough bandwidth to allow a proper audio/video stream. After the video conference Steve and COWORKER 1 can bring their meeting to a successful ending. COWORKER 1 is added as team member and EXPERT 1 is given the choice whether he wants to become a team member or not. The activity is marked as "ended" and the progress of ACTIVITY 1 is written to the context database. Whenever one of the two colleagues works on ACTIVITY 1 an entry in the context database is made. Therefore it is possible to query a list of co-workers, who are currently busy with ACTIVITY 1. At the end of TASK X Steve has the possibility to estimate the efforts made to complete ACTIVITY 1.

Queries resulting from the use case

- Get the current location of a user.
- Get communication capabilities of a user.
- Get preferred device of communication.
- Get status / status message of a user.
- Get artefact.
- Get users filtered by certain skills / skill profiles.

- Get current device of user.
- Get current network profile of device.
- Get current activity of user.
- Get involvements of user X in activity X.

The context model has to be able to store and retrieve the information requested by these queries. An UML of the context model can be found in Figure 11 on page 60.

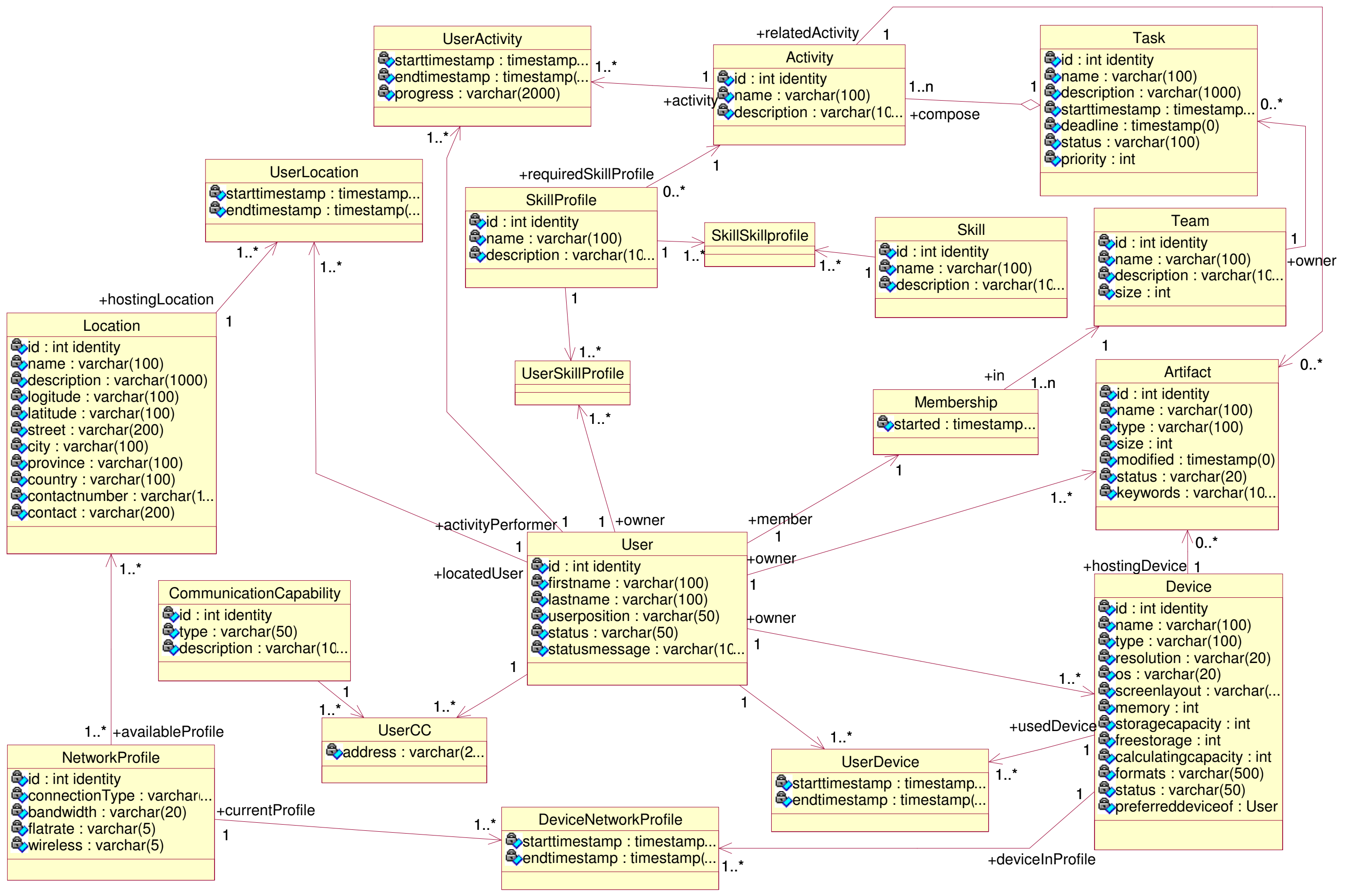


Figure 11: Context Model

4.3 Architecture

There are a few requirements which have to be satisfied to develop a framework for constructing context-aware applications, targeted for mobile devices. The most important requirements are:

- Lightweightness: The framework has to be kept lightweight to meet the limited capabilities of the targeted platform.
- Storage of context information: At least one device in the network has to act as “server” and thus has to store all context information.
- Hosting of context information: The devices on the network have to be able to communicate with each other for information exchange.
- Advertising and discovery: Due to the decentralized architecture, mobile devices need to be aware of each other and of the services one or more of them is/are providing.
- Integration of external sensors: The framework should have the possibility to incorporate external sensors like GPS or ultrasouns sensors.

To meet the “Lightweightness” requirement, the proposed architecture is based on the OSGi Architecture, which has been designed to run on the targeted devices. The Context Store component, as the name indicates, stores and provides context information. The service provider makes the Context Store accessible over the network using standard

Web services. OSGi also comprises standard services that the service provider indirectly makes use of. As we can see in Figure 12 the service provider uses the AMIGO toolkit to establish a Web service, whereas AMIGO uses the bundled HTTP server of the OSGi framework. The Service Advertiser is responsible for the announcement of the Context Store on the network. The counterpart of this is the Service Discovery component, which does exactly the opposite, namely search for Context Stores that are accessible online.

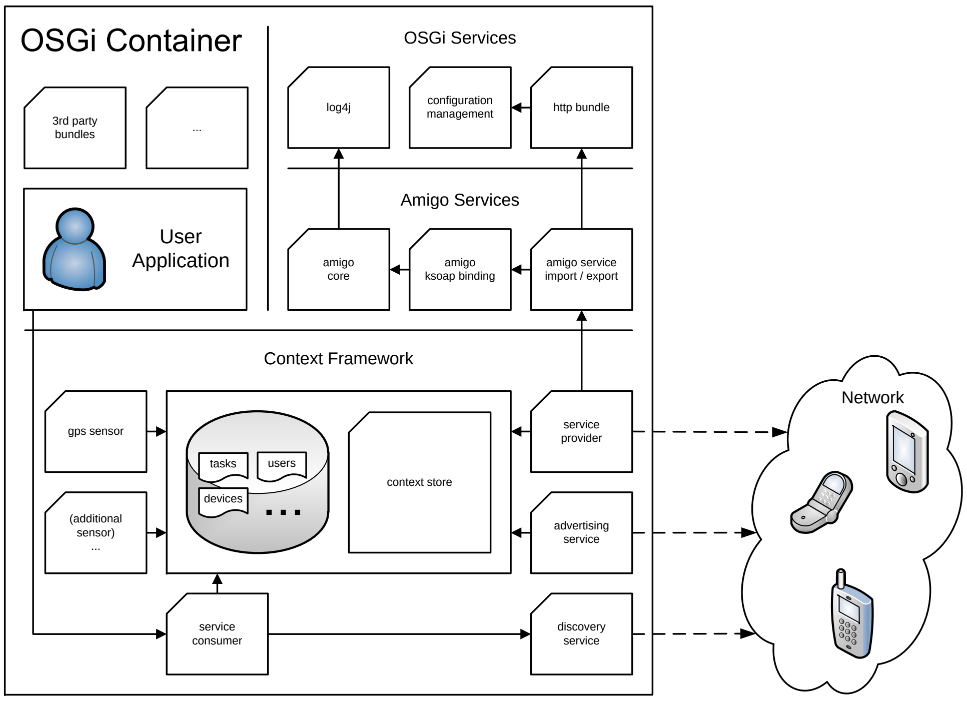


Figure 12: Architecture: Server configuration

Components

To encapsulate the functionality of the framework the OSGi bundle application design paradigm has been used and the functionality of the

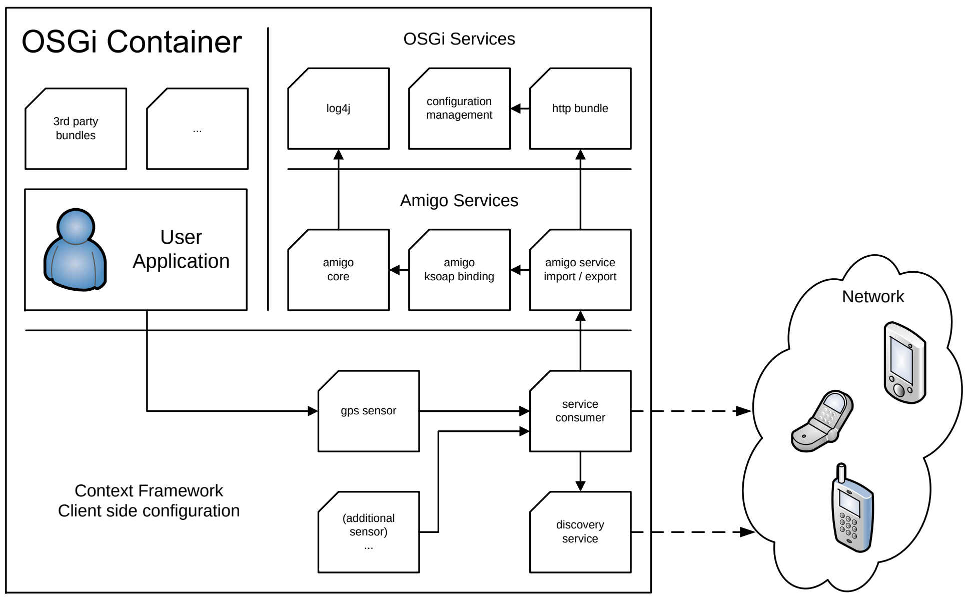


Figure 13: Architecture: Client configuration

framework has been devided into 8 OSGi bundles. In this section the components, their roles and tasks will be introduced in detail. Figure 14 on page 65 shows the bundle dependencies in detail. The sizes of the bundles give a rough impression of the footprint of the bundle.

cfw-context-store-api

This bundle contains the entities that represent the context information and the interface to the Context Store, which is implemented by the cfw-context-store bundle. As the name already implicates, this bundle has been introduced to make other bundles aware of how to access the Context Store bundle. The entities are also needed here, because only full entities and arrays of entities respectively are transferred between the Context Store and a client. This API bundle is a simple

Bundle	Function
Context Store	Stores and provides context information
Service Discovery	Searches for Context Stores
Service Advertiser	Advertises a Context Store for remote access
Service Provider	Makes a Context Store available for others via SOAP-based Web services
Service Consumer	Required to access a Context Store either remotely or locally (hides the type of the service)
User application	Communicates with the framework using OSGi services
GPS	Example for an external Sensor connected to the framework. Also other sensors like ultrasound or bluetooth are imaginable.

Table 1: Components of the framework

"library"-bundle without any functionality. The only thing it does is exporting two packages to the OSGi framework and thus has no activator.

cfw-context-store

The Context Store bundle is responsible for storing the context information submitted by the local or remote devices. It consists of three components

- Database

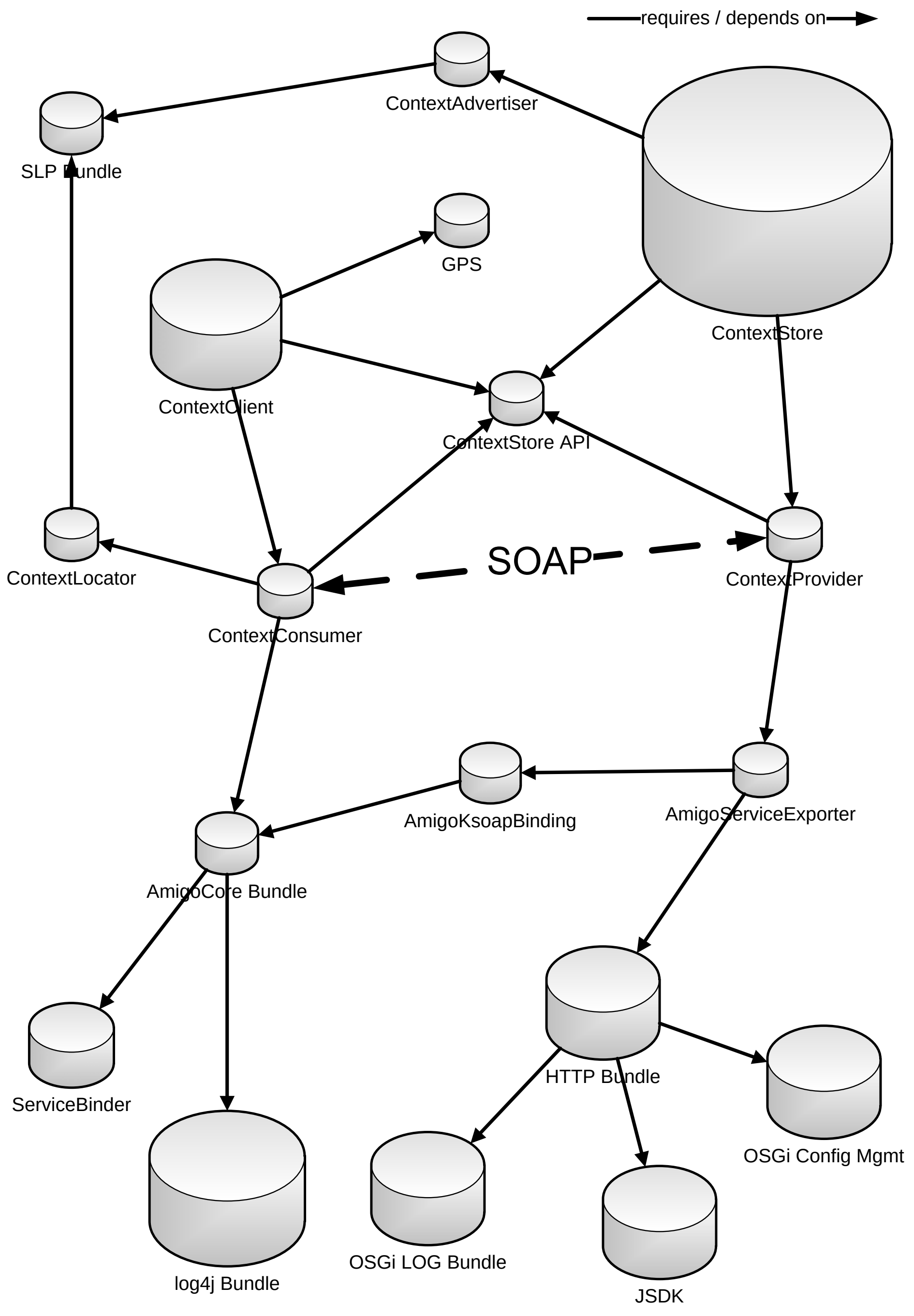


Figure 14: Bundle dependencies

- Persistence Engine
- API Implementation

The context database is built atop hsqldb [7], a 100% pure Java relational database, which is very lighweight and efficient. It is originally written for Java 2 SE, but there also exists a CDC port of the library [8]. The only difference between those packages is, that the original version uses the JDBC driver API to connect to the database. This way is not supported by CDC-JDBC, which caused the developers to implement an alternative. The acquisition of a data source is well documented in the package. The hsqldb library in version 1.8.0 has a footprint of 495KB.

As mentionend in Section 2.1, the CDC profile requires a optional JDBC package to be able to handle databases. In this case, I used the reference JDBC implementation for CDC (JSR169) of Sun [11], which has to be built and included in the bootclasspath of the JVM.

The database is accessed via relational persistence engine called Mr. Persister [19]. The use of this engine leverages the database access dramatically, because objects can be stored and updated easily without forming custom SQL queries. I have also written a database abstraction class, in which I made use of the reflection API to create generic methods for easy inserting, updating and getting objects in/out of the database.

Due to the use of the above mentioned libraries and the database abstraction, the implementation of the Context Store API could be kept very simple and concise. Most of the methods implemented are intended to deliver context information out of the database. They basically consist of a SQL query and a call to the database API. An example would be

CS.getArtifactsOfUser(int userId);

This method retrieves all artifacts of a user with User ID userId in form of an Array of Artifact-objects, provided by the API bundle. Here only a few methods have been implemented, because it is rather straight forward and the interface and implementation can be extended easily. The implementation done by me only satisfies demonstration purposes.

On startup, this bundle creates an instance of the Context Store implementation and registers it as an OSGi Service. If a cfw-context-provider bundle is installed and started, the Context Store bundle uses its service, to make the Context Store instance accessible for remote devices. If an advertiser service is present, it also announces the service on the network using SLP.

cfw-context-provider

This bundle is the implementation of the Service Provider and slightly dropping out of the scheme, because of the architecture of AMIGO. This bundle uses the Gravity Service Binder [6], because the AMIGO Service exporter requires it to do so. The only purpose of this bundle is to export a locally present ContextStore OSGi service as an AMIGO service. If no local service is present, the bundle is inactive, because in this case no Context Store is present, which can be exposed as a Web Service.

cfw-context-advertiser

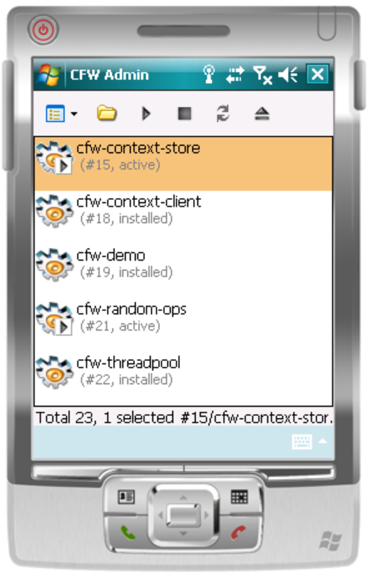
This part of the framework represents the Service Advertiser registers a service to the OSGi service pool, which is used to advertise a Context Store on the network. The bundle makes use of the jSLP bundle provided by the ETHZ [13] make other devices connected to the network aware of a running Context Store. It gets the host name, the http-port of the OSGi framework and forms an endpoint URL, which is published.

cfw-context-locator

The context locator is the implementation of the Discovery Service and has been designed to be the counterpart to the above mentioned context advertiser. This bundle also makes use of the jSLP bundle to find Context Stores. It registeres a locator OSGi service, which can be used by other bundles to retrieve those endpoint URLs.

cfw-gui

This bundle is based on the Knopflerfish AWT framework desktop bundle and has been modified to check the status and take control over the context framework. It can be used to start and stop components of the framework and has mainly been developed for testing purposes. It shows only components of the context framework, and leaves the other bundles installed in the OSGi container untouched (see screenshot in Figure 15).



cfw-context-store
(#15, active)
cfw-context-client
(#18, installed)
cfw-demo
(#19, installed)
cfw-random-ops
(#21, active)
cfw-threadpool
(#22, installed)
Total 23, 1 selected #15/cfw-context-stor.

Figure 15: Administration Bundle

cfw-random-ops

The random operations bundle has been developed to test the stability of the Context Store. It demonstrates how many requests per interval the Context Store can handle and if there are any collisions if several concurrent requests are received. The bundle also shows problems like crashes or exceptions, in case the interval is shorter than the total request time and if the requests are queued properly. The bundle uses an algorithm based on fibonacci numbers, where the interval I are calculated using the equation

$$I = \frac{1}{f} * 1000ms$$

where f is sequentially taken out of the set of the first 20 fibonacci numbers. Five requests are sent using the current interval, then the next fibonacci number is used. This ensures that the intervals are getting shorter, and the load on the Context Store gets heavier. This bundle was used during the performance evaluation, to calculate the typical request times.

cfw-threadpool

This bundle is based on an existing threadpool implementation taken from the JGrinder project [12] and is designed to be a kind of a stress test for the Context Store. On startup, the bundle spawns up a customizable number of threads, which concurrently send a request to the Context Store. Receiving multiple requests at the same time shows whether the Context Store is able to handle concurrent requests without failure. See evaluation for results.

cfw-demo

To demonstrate the usage of the framework, this sample application has been implemented. It basically is an activity manager, which lists all the tasks the team of the mobile device user is responsible for. The user can now take a closer look at the activities of the task, modify details of them or add new activities to the task. At the startup of the bundle, the connection to a Context Store is established, either if it is installed locally or situated remotely and the list of tasks is fetched. This process is hidden from the user. As already mentioned, this bundle has been written to show how to fit the components of the framework together and to view the SOAP messages, which are sent to and received from the Context Store using the TCP Monitor from the Apache Axis package. Here are a few screenshots of the demonstration bundle:

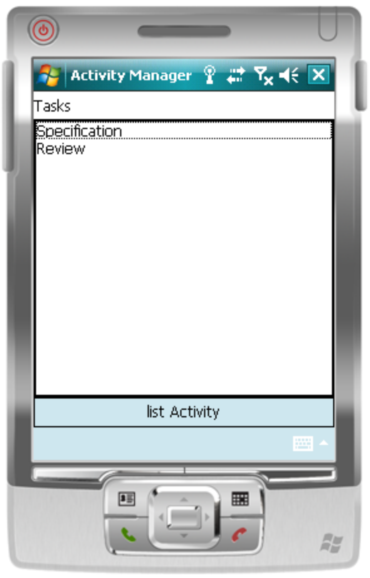


Figure 16: The Activity Manager on a Pocket PC

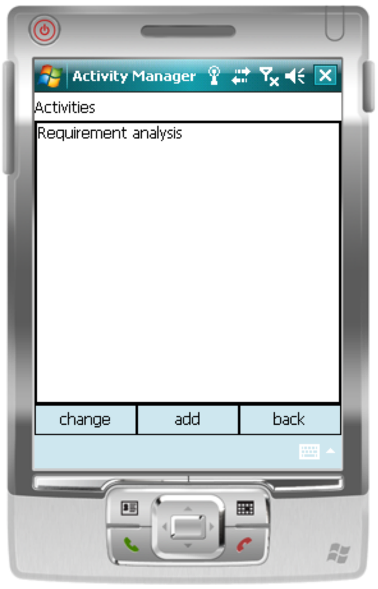


Figure 17: The Activity List on a Pocket PC

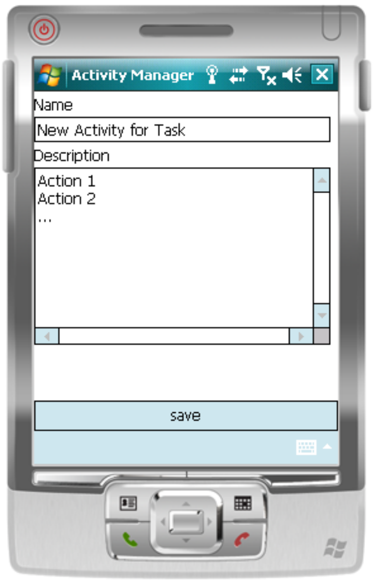


Figure 18: Add Activity

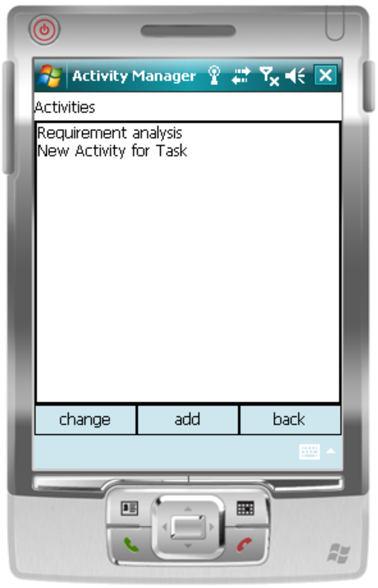
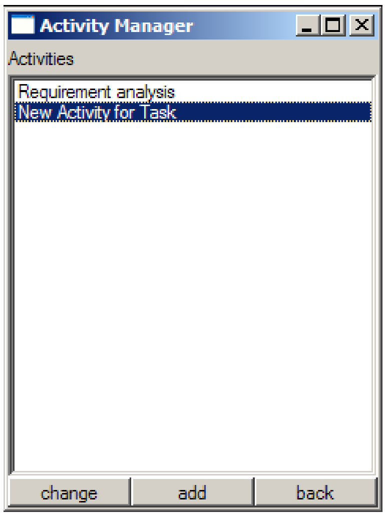


Figure 19: Task with new Activity

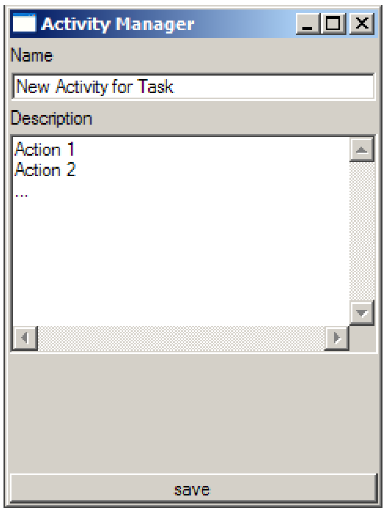
Server and Client Configuration

The framework is designed modular using the OSGi bundle mechanism. Functionality has been encapsulated in components, which have



Requirement analysis
New Activity for Task

Figure 20: Activity List on the remote peer



Name: New Activity for Task

Description:
Action 1
Action 2
...

Figure 21: Details of the new activity on the remote peer

been kept as small as possible, to ensure easy management and greatest flexibility possible.

Due to this structure, a system running the framework has to be configured according to the tasks it has to fulfill. The framework is designed to run in two main configurations: A “server” configuration, providing and collecting context information and a “client” configuration sending and retrieving context information. Of course those configurations can be modified adding components (e.g. additional sensors), but the vital parts of the framework and the dependencies have to be installed.

Server Configuration

The server configuration is an OSGi container running at least the cfw-context-store bundle, the cfw-context-provider bundle, the cfw-advertiser bundle and their dependencies. This configuration starts up the Context Store, which is registered locally as an OSGi service and registered in the Knopflerfish HTTP server as an AMIGO Service. The cfw-advertiser bundle publishes the endpoint of the AMIGO service and ensures that it is found via SLP.

For administration purposes, this configuration can be extended with the cfw-gui bundle, which allows certain service to be started and stopped at runtime.

Client Configuration

The client configuration is the counterpart to the above mentioned server configuration. It consists of the cfw-context-locator bundle, the cfw-context-consumer bundle and their dependencies. This configuration enables a potential developer to find and access any device running a server configuration on the net. The consumption of a ContextStore service is very easy, because the ContextStore can be acquired with a simple method call. All the internas like discovery, binding etc. are hidden from the developer.

For demonstration purposes and as a how-to tutorial, I have written four bundles, which show the usage of the framework. Hence, the usage of the administration bundle (cfw-gui) is recommended in the client configuration, to selectively start the demonstration bundles.

Due to the flexibility of the framework, client and server configurations can be combined. Both can run on a single device. The framework recognises this case and establishes communication between the server and the client using the OSGi services. Communicating over HTTP using AMIGO would not make sense here, because it is much slower and eats up resources of the device.

4.4 Design and Implementation

4.4.1 Developing Services for Mobile Devices

Platform

The choice of the platform to set the framework atop was very important. The decision was made taking the following aspects into account:

- Lightweightness
- Platform-Independency
- Modularity

Considering various possibilities, the framework is built atop the Knopflerfish OSGi implementation [16], which is a very lightweight one and made available as open source. OSGi also has the advantage, that it requires a Java Virtual Machine, which is available for many embedded and lightweight mobile devices. Remember that nearly all mobile phones today feature at least J2ME. So Java perfectly satisfies the second point on the list, the platform-independency. As mentioned in the technology review section, applications for OSGi are packed as seperat bundles, which enables the developer to encapsulate different types of functionality.

Apache ANT

Apache Ant is a build tool based on JAVA. It is similar to “make” on UNIX based systems. Build files are written using XML, which makes it easy to define complex dependencies and configurations of build targets. Another advantage of Ant is, that it is platform independent (because based on Java). For this project it is required to build the AMIGO stubs, which are generated using an external Java library, that is called during the Ant task.

Managing Service Dependencies in Deployment

As mentioned in the OSGi Services section above, managing services and their dependencies can be quite laborious when the number used services increase (Service A requires B, but B requires C and D etc). The Gravity Service Binder is an attempt to facilitate the OSGi service management. Service dependencies don't have to be managed manually anymore, the dependencies can be defined in an XML configuration file (see example on page 96) and the rest is done by the service binder. This reduces the amount of code necessary for dependency management dramatically.

Knopflerfish Bundle IDE for Eclipse

The developers of the Knopflerfish framework provide bundle IDE for Eclipse, which alleviates the development of bundles for the OSGi framework. It overtakes tasks like checking dependencies with other bundles and packing all resources of a bundle to a JAR. It also features a manifest editor for OSGi bundles with auto completion functions, which enable the user to create sophisticated and consistent bundle manifests. Finally, the plugin integrates the Knopflerfish OSGi Framework into Eclipse, allowing to set up special startup configurations with a user interface to check bundle dependencies before runtime.

4.4.2 Data Persistency

JSR169 - JDBC for CDC

The Connected Device Configuration Profile requires a kind of "database layer" to be able to operate with hsqlDB. This is done by JSR169, which specifies a JDBC library for the CDC profile. JDBC (Java Database Connectivity) is a uniform API to different kinds of databases, but especially designed for use with relational databases.

In this project the reference implementation from Sun [14] is used.

hsqlDB - 100% pure Java Database

hsqlDB [7] is a very lightweight relational database engine written in Java, which seemed to be the optimal choice for storing the context information on a mobile device. The developers of this open source software offer also many special versions of the product (e.g. a version for embedded devices, applets, servers etc.). There is also a special CDC version available, which fitted perfectly in my setup.

Relational Persistence

To facilitate the data management I used Mr Persister [19] as a relational persistence engine. With this tool, Java objects can be mapped to table records and vice versa. This reduces the coding efforts storing or fetching data from the database to a minimum, because some odd tasks like forming custom sql queries for storing or updating records are not necessary anymore. Mr Persister™is also very lightweight and perfectly compatible with hsqlDB.

5 Evaluation

5.1 Testsetup

5.1.1 Configurations

For testing purposes the two configurations mentioned above were used. A server configuration hosting the Context Store, which consisted of

- cfw-context-store
- cfw-context-store-api
- cfw-context-provider
- cfw-context-advertiser
- cfw-context-consumer
- cfw-context-locator
- cfw-gui
- and dependencies

The client configuration requires nearly the same components but of course the Context Store is not included. The GUI bundle is also not needed.

5.1.2 Footprint of the Framework

The files needed on the mobile devices occupied around 1,86 MB. This does not include the space required for the IBM J9 JVM, which occupies again 14,3 MB of memory including the libraries (db-enabler, jdbc and miscellaneous classes) are required by the extension class loader to initialize the JVM properly for execution of the framework.

The largest component in the framework is the Context Store itself. The bundle occupies 645 KB as it contains the database engine, the relational persistence engine and the implementation of the Context Store interface. This bundle is not needed in the client configuration because it is accessed over the network on a remote device. In this case, the footprint can be reduced by this amount of bytes. However, if we take a look at the space solely the IBM J9 JVM occupies, these 645 KB do not make a big difference.

Detailed footprint information can be found in the table in the Appendix on page 111. Also the Figure 14 on page 65 gives an impression of the bundle sizes.

5.1.3 Testsetup

A client configuration was deployed on a PC (Intel Core 2 Quad, 4x 2,4GHz, 3GB system memory), which is connected to a 100MBit switch. The server configuration was deployed on a HP Mobile Messenger (iPAQ hw6915), which has been associated with a Netgear WG601v1 802.11g access point. The access point was also connected to the switch as you can see in figure 22. The average ping time between desktop and the HP Mobile Messenger was typically around 5-6ms.

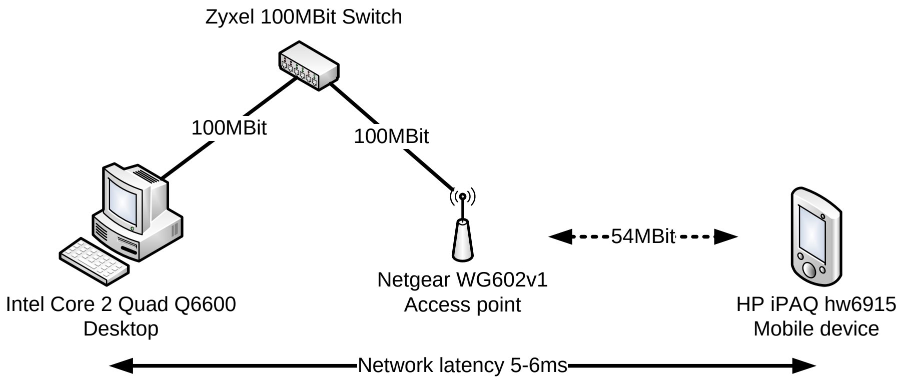


Figure 22: Test setup

The Context Store was started on the mobile device, with advertising enabled. Next, the random operation bundle was started on the deskop computer and the client initialized the discovery mechanism properly. The Context Store was found instantly and the random operation bundle began to send requests to the Context Store.

5.2 Performance study

This performance study gives an impression of the behaviour of the framework on an average Pocket PC, which it is targeted to run on. An additional assumption is, that the framework is used in small and medium nimble teams respectively, which in average have around 10 members. Depending on the results, we will be able to estimate, if it is possible to develop responsive applications based on the framework.

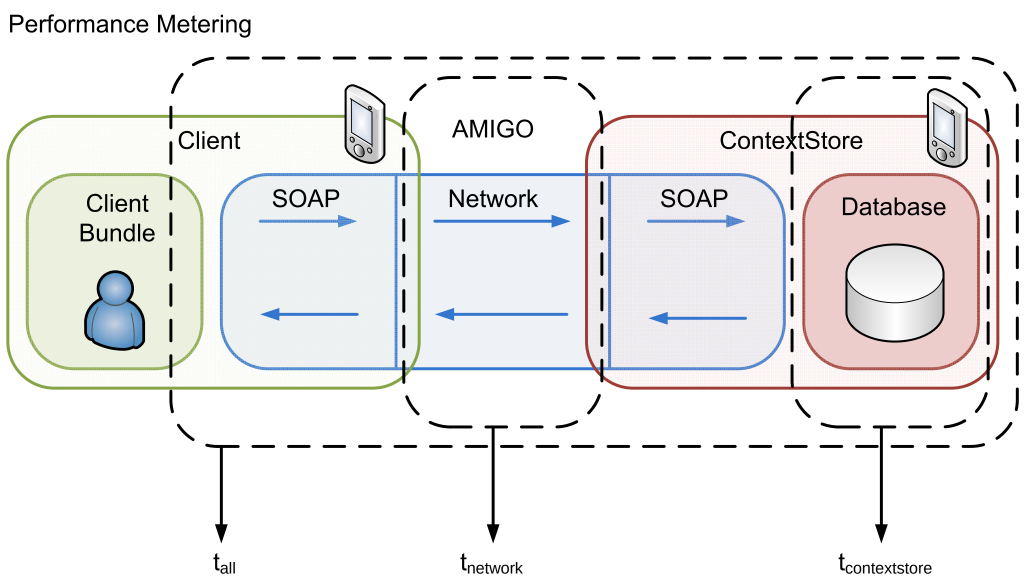


Figure 23: Time intervals for performance metering

Averaging request time 10 to 40 we can see that the total time from a request to the Context Store until a response from the Context Store is around 657ms. This time includes the time needed for the SOAP envelope generation at client side, the time for the SOAP parsing at Context Store side and vice versa. We also have to add the network latency, which in my tests was around 2,5-3ms in one direction. The average access time of the Context Store was around 300ms. We can now calculate the average time that Amigo needs for the SOAP envelope generation and parsing and the mapping of Java objects to XML and the other way round using

$$t_{soapgen} + t_{soapparse} + t_{mapping} = \frac{t_{total} - t_{contextstore} - t_{network}}{2}.$$

On the client side amigo needs to generate a request ($t_{soapgen}$) and send it over the network. The Context Store has to parse the request ($t_{soapparse}$) and map the contents to Java objects ($t_{mapping}$). This is done by Amigo and therefore not measurable in detail. If we assume, that the request and response consumes approximately the same amount of time, we can estimate the time Amigo needs to do its work. Taking the network latency of 3ms per direction into account, Amigo needs in average 352ms exchange request and response.

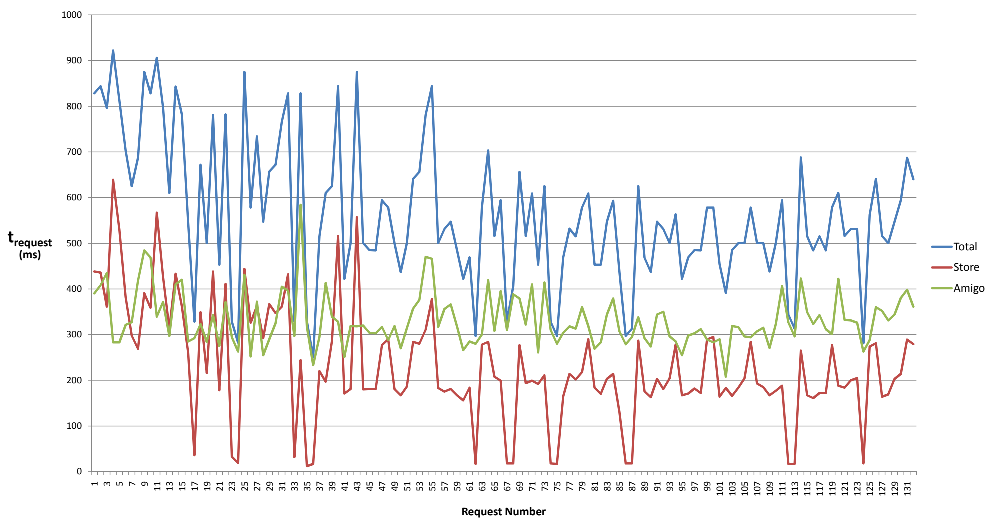


Figure 24: Performance

The chart was cut off at the beginning, because the first request always required around 8 seconds to complete. When the first request arrives, the database connection is established and the persistence engine is initialized. This outlier is therefore not displayed in the diagram. When taking a look at the rest of the chart, we can see, that the average total request time lies a little above 500ms. Exact values can be found in Table 5.2.

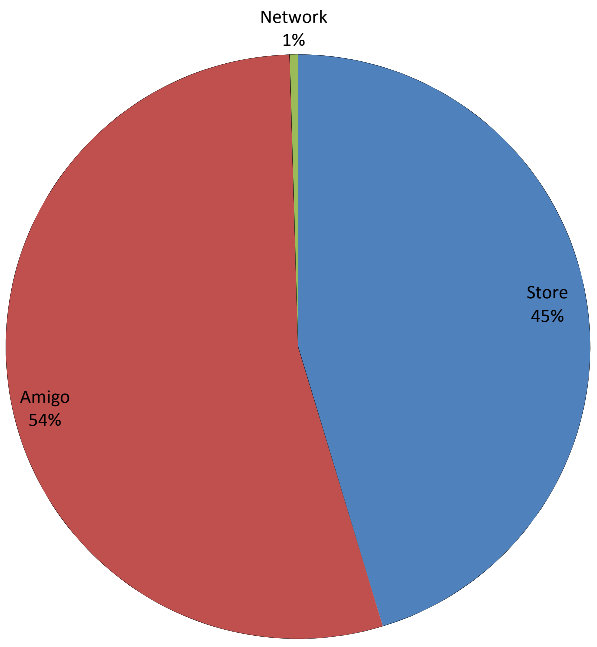


Figure 25: Average Performance

	AVG	SD	MIN	MAX
Total	657 ms	199.29	250 ms	906 ms
Store	299 ms	152.02	12 ms	567 ms
Amigo	358 ms	78.82	233 ms	584 ms
Network	2.73 ms	0.25	2.5 ms	3 ms
Request rate	1.81	0.70	1.10	3.55

Table 2: Average, minimum and maximum request time

5.2.1 Stresstest

Using the cfw-threadpool bundle multiple virtually simultaneous requests can be fired at the Context Store. My tests have proven that the Context Store has no problems dealing up to 100 and maybe even more of simultaneous requests. This implicates that the built-in HTTP server of the Knopflerfish framework seems to work really stable and that the requests are queued properly. The limiting factor here is the execution time on the Context Store side, because the requests cannot be processed all at the same time. The stresstest was stopped at 100 requests, because it is considered unlikely that in the target field of the framework (nimble teams with 10 team member average) the Context Store receives more than 100 simulaneous requests.

5.2.2 Conclusion

The performance study has shown, that the Context Store is able to process at least 1 and at maximum 3 requests per second from other mobile devices, which can be considered to be acceptable for a small or medium mobile team. Of course it depends on the type of application, that is built atop the framework, but for basic information exchange this is enough (e. g. nobody can change or add activities in less than one second).

As expected, the communication overhead resulting from the use of Web services is considerable. We can say that at least half of the request time is needed by AMIGO to generate and parse the soap messages, which has to be done twice in each case. But Web services do not only cause large overhead in the world of lightweight and mobile devices, this is also an issue when building applications for “normal” devices, like desktop PCs and servers. However, they have many advantages like interoperability, ubiquity and loose coupling.

5.3 Implementation Aspects

5.3.1 AMIGO

Popular SOAP toolkits, which are available as open source projects for Java and running with the Personal Profile, have been considered in this work. Most of them require the programmer to create XML configurations, that describe the mapping between the methods and the exposed Web service. This description has to include the method name, the parameters and the return values to properly map an interface to a Web Service. Amigo has a different approach as it tries to generate the service from the class definition by making extensive use of the Java Reflection API, which tremendously facilitates the development of a complex interface.

But the other side of the coin is, that the usage tutorial provided by the developers is insufficient when dealing with complex objects or data types. The examples provided work well, but they are simple and not practicable in a “real” application. The handling of various datatypes is very unfirm, maybe because Amigo fully relies on the kSOAP2 project, which has proven not to be rock solid during my tests. It took me weeks to figure out a way to handle the Amigo libraries, which easily could have been avoided with proper documentation.

There are also some features, which could not be used, because of the lack of documentation. The performance of the dynamic stub generation would have been very interesting, but I was not able to get it working. This would even more alleviate the development process and make it possible to change the interface at runtime.

5.3.2 OSGi

OSGi for has proven to be a very sophisticated concept and the implementations available, especially the Knopflerfish framework, work like a charm. Due to the bundle concept, the developer is automatically encouraged to encapsulate application functionality. For a framework development, this is very useful, because components can be combined according to the users needs.

A drawback is the slow startup of the framework, but this issue is more Java than OSGi related. The bootstrapping of the JVM on an up-to-date Pocket PC still takes a couple of seconds and not till then the execution of the application is started. Another point is, that the security features of OSGi can make it tricky to fulfill the developers requirements.

5.3.3 SLP

The SLP implementation [13] used for this project is working fine in the latest version of the distribution. In this version the SocketTimeoutException bug was fixed, which caused the bundle to stop working after the first discovery. The documentation is good and the tutorial allows to quickly pick up the usage of the bundle.

6 Summary and Conclusion

This thesis gave an overview about the current status of Web services in mobile computing. The survey has shown, that the hosting of SOAP based Web services on mobile lightweight devices is possible but there is still need for sophisticated toolkits. Currently available libraries suffer from problems like instability and require lots of programming and configuration efforts to build Web service based applications. Sophisticated Web service toolkits like Apache Axis cannot be run on mobile devices, because they require too much resources. The platforms needed for deployment are often not available for mobile devices.

This work also presented a proof of concept implementation of a SOA based collaboration framework, which enables the developer to host context information on a lightweight device using Web services. The framework is based on OSGi and thus following the Service Oriented Architecture paradigm.

The deployment of Web services on mobile devices is still linked with several difficulties. But although there are still problems hosting Web services on mobile devices today due to several issues like limited resources, lack of processing power and memory, limited wireless network throughput, it is likely that those problems are solved in the near future through the expansion of mobile networks and the constant extension of the capabilities of mobile devices.

A Knopflerfish on a PPC using J9

A.1 The Knopflerfish Framework

Setting up Knopflerfish OSGi on a Pocket PC requires a desktop pc or laptop and of course: a Pocket PC. Those, who do not own or do not have access to a Pocket PC, can use the Microsoft Device Emulator bundled with the Microsoft Visual Studio. Maybe this is the better way to play around with the framework, because the virtual devices are highly configurable, can be connected to the local network and internet without spending any money for WiFi and LAN adapters. Memory of the device can also be extended, which also helps to avoid resource bottle necks. For communication purposes with the host machine a shared directory can be configured, that is accessible through the storage card in the Pocket PC emulator.

There are several OSGi implementations on the web. One of them is Knopflerfish OSGi, which we chose to use for our purposes. Although the framework seems to be ready to install, many problems and pitfalls arise when deploying it on a Pocket PC running Windows Mobile 2003SE.

A.2 Java Runtime Environment

Since there is no native support for Java in WM2003 an appropriate Java Virtual Machine has to be installed on the Pocket PC. There are several competitors on the market for example NSIcom CrEme, SavaJE OS, Sun Personal Java, Blackdown J2RE, Esemertec Jeode, IBM J9 JVM etc. For my project the JVM at least had to support class loading and Pocket PC, that's why most of the competitors where not usable. Only the NSIcom CrEme and the IBM J9 JVM support the Personal profile (PPRO), which is needed for class loading. Additionally the IBM J9 JVM is the cheaper choice, which was the reason for the decision to use it in this project.

The IBM WebSphere Studio Device Developer includes the WebSphere Everyplace Micro Environment (WEME), which consists of two device profiles (PPRO10 and MIDP20). Since the MIDP profile does not support class loading we have to install the Personal Profile, which requires a lot more memory (around 8mb), but is also more powerful. The installation normally completes without any troubles, but it gets very challenging when trying to run a java program.

A.3 Executing the JVM

First of all, the Pocket PC only offers a hidden possibility to execute applications. It's the well known "Run" prompt, which we all have seen on the popular Microsoft Windows operating system. To display this command one has to press and hold main enter button on the Pocket PC and tap on the clock with the stylus. In this prompt the JVM can be executed. In fact, this method is not very satisfying, because the field is very small, and the execution commands of Java programs typically become very long.

The next possibility to execute the JVM is a command prompt similar to the DOS command or unix/linux shell. SymbolicTools provides such a tool, called PocketConsole, which is nearly a perfect port of the Windows NT command shell and much more comfortable than the “Run” command line.

Another alternative for the two mentioned above is to connect the Pocket PC via active sync to a desktop pc and create a shortcut file, executing the JVM with all necessary parameters. The shortcut can be created using the copy and paste method on the Pocket PC. If the emulator is used, the best way of sharing the shortcut with the host computer, is to but it on the shared space (Storage Card). This method seems to be the best possibility, because typing on the desktop pc is much easier than tapping around on the Pocket PC touch screen, but it has some perfidies, which will be mentioned later. Creating a shortcut with the Pocket Explorer produces a file with the extension *.lnk, which is simply a text file containing the execution string, something like this:

```
38#”\Program Files\J9\PPRO10\bin\j9.exe”
```

The number at the beginning of this string indicates the length of the following command. In case the command is edited and thus becoming longer than the original, this indication also has to be modified to suit the new strings length or must be set even higher. IBM [23] suggests setting it to 255, but tests have proven, that the command is not executed correctly if the length exceeds this boundary. As a solution for this problem I recommend to use the command shell.

A.4 Preparing J9 for Knopflerfish

To start the J9 JVM we have to browse to ”\Program Files\J9\PPRO10\bin”

where we find two executables (”j9.exe” and ”j9w.exe”). The first one is the console version of the JVM and the second starts the JVM without opening a window. For testing and debugging purposes the usage of the first version is strongly recommended, because most likely the application will not run the first time. A strange behaviour of the installed JVM is that it tries to start with the never installed Foundation Profile, which can only be avoided by adding the switch “-jcl:ppro10”, which forces the JVM to start with the right profile.

It should also be mentioned, that the default installation does not install all necessary libraries for running the Knopflerfish OSGi. If we try to run the jar package of the framework, we get a ZipException, which points to the fact that we have to copy the j9zlib22.dll library to the bin directory. This extension empowers the JVM to handle compressed jar archives.

The next step is to obtain the current version of the Knopflerfish framework from their website [16]. Depending on the storage capacity of the device we are able to choose between different packages of the framework. The bundles can be obtained and installed separately. For our purposes we chose to download the full package because the virtual storage card in the emulator has nearly unlimited capacity due to the fact that it’s only a shared directory on the host hard disk.

A.5 Setting up Knopflerfish

Running the Knopflerfish framework is not possible without some modifications of the configuration files. The main problem is that the Pocket PC does not support a "current directory", which is the directory a command is called from. Knopflerfish searches for its configuration files in the current directory by default, which results in an error if the framework is not called from the root of the fixed memory on the Pocket PC. The fwdir, which is the directory where the framework stores persistent bundle information, is also stored there. Although it offers an option to specify which configuration files to use, the framework produced an error during several tests. A solution is to put the properly configured init.xargs, which holds the commands and settings for initialisation of the Knopflerfish OSGi framework and restart.xargs, required for restarting the framework using the previous settings and persistent data in the fwdir, in the root of the Pocket PCs fixed memory to avoid all those problems. The props.xargs fire holds the manually configured system properties of the framework. It is of paramount importance that the options in init.xargs and restart.xargs reference to the right location of this file, if it is not stored in the same directory (in our example the root of the fixed memory of Pocket PC).

The framework is able to start on the Pocket PC under J9 Personal Profile with the default bundles, but the desktop interface bundle has to be excluded. It is not compatible with the AWT libraries of the J9. This can be done by simple commenting the startup command in the configuration file.

Starting the framework for the first time, invokes the installation of the bundles configured in the init.xargs file and the bootstrapping. In our tests we used the above mentioned method of executing the startup command per shortcut, which seemed to work fine, but after bootstrapping the framework exited with a dubious error message “Error: Unknown Option:”. This leads to the assumption that invoking the J9 per shortcut passes an extra argument, which is not recognized by the framework. Therefore the less user friendly command shell must be used to initialize and start Knopflerfish.

A.6 Programming OSGi bundles

The Knopflerfish community provides an excellent eclipse plugin, which is very helpful for creating OSGi bundles for the framework. It can be obtained via web update of the eclipse platform [17]. The plugin enables the user to integrate the two possibly most popular OSGi frameworks Oscar and Knopflerfish with the corresponding bundle repositories.

The plugin allows creating runtime configurations very easily, the (de) activation of every single bundle and always points out the bundle dependencies. Execution environment profiles are selectable e.g. CDC-1.0/Foundation-1.0, which makes the testing of the bundle for the target platform very easy. The GUI access to the bundles manifest files and the automatic jar generation definitely one of the greatest advantages. The usage of the plugin for bundle development is highly recommended.

B Code Listings

B.1 CFW-API-Bundle ANT build file

Listing 6: ANT build file to generate stubs

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<project name="amigo_custom_bundle" basedir="." default="
   all">

  <property name="out.dir" value="out" />
  <property name="src.dir" value="src" />
  <property name="bundle.name" value="cfw-context-store-api
     -1.0.0.jar" />
  <property name="req.libs.dir" value="D:/java/osgi/_amigo/
     libs" />
  <property file="build.properties" />

  <path id="classpath">
    <fileset dir="${req.libs.dir}">
      <include name="*.jar" />
    </fileset>
    <dirset dir="D:/java/osgi/cfw-context-store-api/out" />
  </path>

  <taskdef resource="macrodefs.xml" classpathref="classpath
     " />

  <target name="all" depends="bundlification" />

  <target name="init">
    <delete dir="${out.dir}" />
    <mkdir dir="${out.dir}" />
  </target>

  <target name="compile" depends="init">
    <javac srcdir="${src.dir}" destdir="${out.dir}" source=
       "1.3" target="1.4">
      <classpath refid="classpath" />
      <include name="**/*.java" />
    </javac>
  </target>

  <target name="java2stub" depends="compile">
    <java2stub
      type="class"
      sourcepath="${src.dir}"
      genericStub="${genericStub}"
      interfaces="${amigoServices}"
      destdir="${out.dir}"
      velocityDesc="${velocityDescription}"
      docletpathref="classpath"
      classpath="${out.dir}"
    />
  </target>

  <target name="bundlification" depends="compile,java2stub"
     >
<!--    <copy file="metadata.xml" todir="${out.dir}" /> -->
    <jar manifest="bundle.manifest" destfile="${out.dir}/${
       bundle.name}" basedir="${out.dir}" />
    <!--<delete file="${out.dir}/metadata.xml" />-->
  </target>
</project>
```

B.2 Service Binder Configuration

Listing 7: Example of a Service Binder Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<bundle>
  <component class="org.simpleclient.impl.ServiceImpl">
    <provides service="org.simpleclient.interfaces.
       SimpleClientServiceA"/>
    <provides service="org.simpleclient.interfaces.
       SimpleClientServiceB"/>
    <property name="provider" value="Beanome.org" type="
       string"/>
    <requires
        service="org.simpleservice.interfaces.SimpleService
           "
        filter="(version=*)"
        cardinality="1..n"
        policy="static"
        bind-method="setServiceReference"
        unbind-method="unsetServiceReference"
    />
  </component>
</bundle>
```

B.3 Context Model SQL

Listing 8: Database script of the context model

```
create table user (
  id int identity,
  firstname varchar(100) not null,
  lastname varchar(100) not null,
  userposition varchar(100) default 'null',
  status varchar(50) default 'offline',
  statusmessage varchar(1000) default 'null'
);


create table communicationcapability (
  id int identity,
  type varchar(50),
  description varchar(1000)
);


create table usercommunicationcapability (
  user int not null,
  communicationcapability int not null,
  address varchar(200) not null,
  foreign key (user) references user(id) on delete cascade
     on update cascade,
  foreign key (communicationcapability) references
     communicationcapability(id) on delete cascade on
     update cascade,
  primary key (user, communicationcapability)
);


create table team (
  id int identity,
  name varchar(100) not null,
  description varchar(1000) default null,
  size int default 1
);
```

```


create table membership (
  started timestamp(0) default current_timestamp,
  owner int not null,
  team int not null,
  foreign key (owner) references user(id) on delete cascade
      on update cascade,
  foreign key (team) references team(id) on delete cascade
     on update cascade,
  primary key (owner, team)
);


create table skill (
  id int identity,
  name varchar(100) not null,
  description varchar(1000) default 'null'
);


create table skillprofile (
  id int identity,
  name varchar(100),
  description varchar(1000)
);


create table skillskillprofile (
  skill int,
  skillprofile int,
  foreign key (skill) references skill(id) on update
     cascade on delete cascade,
  foreign key (skillprofile) references skillprofile(id) on
      update cascade on delete cascade,
  primary key (skill, skillprofile)
);


create table userskillprofile (
  owner int,
  skillprofile int,
  foreign key (owner) references user(id) on update cascade
```

```
      on delete cascade,
  foreign key (skillprofile) references skillprofile(id) on
      update cascade on delete cascade,
  primary key (owner, skillprofile)
);


create table task (
  id int identity,
  name varchar(100) not null,
  description varchar(1000) default 'null',
  starttimestamp timestamp(0) default current_timestamp,
  deadline timestamp(0) default null,
  status varchar(100) default 'pending',
  priority int default 5,
  team int,
  foreign key (team) references team(id) on update cascade
);


create table activity (
  id int identity,
  name varchar(100) not null,
  description varchar(1000) default 'null',
  task int,
  foreign key (task) references task(id) on update cascade
);


create table useractivity (
  user int,
  activity int,
  starttimestamp timestamp(0) default current_timestamp,
  endtimestamp timestamp(0) default null,
  progress varchar(2000) default 'null',
  foreign key (user) references user(id) on update cascade,
  foreign key (activity) references activity(id) on update
      cascade
);


create table location (
  id int identity,
  name varchar(100) not null,
  description varchar(1000) default 'null',
  longitude varchar(100) default 'null',
  latitude varchar(100) default 'null',
  street varchar(200) default 'null',
  city varchar(100) default 'null',
  province varchar(100) default 'null',
  country varchar(100) default 'null',
  contactnumber varchar(100) default 'null',
  contact varchar (200)
);


create table userlocation (
  owner int,
  location int,
  starttimestamp timestamp(0) default current_timestamp not
      null,
  endtimestamp timestamp(0) default null,
  foreign key (owner) references user(id) on update cascade
      ,
  foreign key (location) references location(id) on update
      cascade
);


create table device (
  id int identity,
  name varchar (100) not null,
  type varchar (100) default 'null',
  resolution varchar (20) default 'null',
  os varchar(20) default 'null',
  screenlayout varchar(20) default 'null',
  memory int default 0,
  storagecapacity int default 0,
  freestorage int default 0,
  calculatingcapacity int default 0,
  formats varchar(500) default 'null',
  status varchar(100) default 'null',
  preferreddeviceof int default 0,
  owner int,
  foreign key (preferreddeviceof) references user(id),
  foreign key (owner) references user(id) on update cascade
```

```
);


create table artifact (
  id int identity,
  name varchar(100) not null,
  type varchar(100) default 'null',
  size int default 0,
  modified timestamp(0) default current_timestamp,
  status varchar(20) default 'null',
  keywords varchar(1000) default 'null',
  owner int not null,
  device int not null,
  activity int default 0,
  foreign key (owner) references user(id) on update cascade
     ,
  foreign key (device) references device(id) on update
     cascade on delete cascade,
  foreign key (activity) references activity(id) on update
     cascade
);


create table userdevice (
  user int,
  device int,
  starttimestamp timestamp(0) default current_timestamp not
      null,
  endtimestamp timestamp(0) default null,
  foreign key (user) references user(id) on delete cascade
     on update cascade,
  foreign key (device) references device(id) on delete
     cascade on update cascade
);


create table networkprofile (
  id int identity,
  connectiontype varchar (20) default 'null',
  bandwidth varchar (20) default 'null',
  flatrate varchar (5) default 'false',
  wireless varchar (5) default 'false'
);
```

```sql


create table devicenetworkprofile (
  device int,
  networkprofile int,
  starttimestamp timestamp(0) default current_timestamp not
      null,
  endtimestamp timestamp(0) default null,
  foreign key (device) references device(id) on delete
     cascade on update cascade,
  foreign key (networkprofile) references networkprofile(id
     ) on delete cascade on update cascade
);
```

B.4 Test Data Set

Listing 9: Database script of the test data set

```sql
insert into user(id, firstname, lastname, userposition)
   values
  (0, 'null', 'null', 'null');
insert into user(id, firstname, lastname, userposition)
   values
  (1, 'Martin', 'Brown', 'CEO');
insert into user(id, firstname, lastname, userposition)
   values
  (2, 'Erika', 'Smith', 'Leading Assistant');
insert into user(id, firstname, lastname, userposition)
   values
  (3, 'Thomas', 'Schultz', 'Programmer');


insert into communicationcapability values
  (0, 'null', 'null');
insert into communicationcapability values
  (1, 'gsm', 'Standard GSM connection');
insert into communicationcapability values
  (2, '3g', 'UMTS video telephony');
insert into communicationcapability values
  (3, 'phone', 'Standard phone, fixed location');
insert into communicationcapability values
  (4, 'fax', 'Telefax capability');
insert into communicationcapability values
  (5, 'email', 'E-Mail Messaging capability');
```

```
insert into communicationcapability values
  (6, 'icq', 'ICQ, I seek you, instant messaging service');
insert into communicationcapability values
  (7, 'msn', 'MSN, Microsoft Network, instant messaging
     service');
insert into communicationcapability values
  (8, 'gtalk', 'JABBER, Google Talk, instant messaging
     service');


insert into usercommunicationcapability values
  (0, 0, 'null');
insert into usercommunicationcapability values
  (1, 2, '+34 (9399) 8928042 - 342');
insert into usercommunicationcapability values
  (2, 2, '+23 (2355) 8678678');
insert into usercommunicationcapability values
  (3, 2, '+12 (9223) 234556');
insert into usercommunicationcapability values
  (1, 4, 'm.brown@darkside.org');
insert into usercommunicationcapability values
  (2, 4, 'e.smith@cloudnr9.org');
insert into usercommunicationcapability values
  (3, 4, 't.schultz@coderunderground.net');
insert into usercommunicationcapability values
  (1, 7, 'm.brown');
insert into usercommunicationcapability values
  (2, 6, 'e.smith');
insert into usercommunicationcapability values
  (3, 5, '4568765123');


insert into team values
  (0, 'null', 'null', 0);
insert into team values
  (1, 'Workgroup for Project A', 'Experts with knowledge
     required for Project A', 10);
insert into team values
  (2, 'Taskforce B', 'Experts with special skills for time
     critical projects', 233);


insert into membership values
```

```
  (null, 0, 0);
insert into membership values
  (current_timestamp, 1, 2);
insert into membership values
  (current_timestamp, 2, 2);
insert into membership values
  (current_timestamp, 2, 1);
insert into membership values
  (current_timestamp, 3, 1);


insert into skill values
  (0, 'null', 'null');
insert into skill values
  (1, 'Java basics', 'Basics of the Java programming
     language');
insert into skill values
  (2, 'C++ advanced', 'Advanced knowledge of the C++
     programming language');
insert into skill values
  (3, 'Soft skills', 'Special social abilities');


insert into skillprofile values
  (0, 'null', 'null');
insert into skillprofile values
  (1, 'Skillprofile 1', 'Description for Skillprofile 1');
insert into skillprofile values
  (2, 'Skillprofile 2', 'Description for Skillprofile 2');


insert into skillskillprofile values
  (0, 0);
insert into skillskillprofile values
  (1, 1);
insert into skillskillprofile values
  (2, 1);
insert into skillskillprofile values
  (2, 2);
insert into skillskillprofile values
  (3, 2);


```

```
insert into userskillprofile values
  (0,0);
insert into userskillprofile values
  (1,1);
insert into userskillprofile values
  (2,2);
insert into userskillprofile values
  (3,1);
insert into userskillprofile values
  (3,2);


insert into task values
  (0, 'null', 'null', null, null, 'null', 0, 0);
insert into task values
  (1, 'Specification', 'Goal is the complete specification
     of the project', current_timestamp, '2008-01-01
     12:00:00', 'pending', 1, 1);


insert into activity values
  (0, 'null', 'null', 0);
insert into activity values
  (1, 'Requirement analysis', 'Requirment analysis in all
     fields of the project', 0);
insert into activity values
  (2, 'Softwaredesign', 'UML design of the software part',
     0);


insert into useractivity values
  (0, 0, null, null, 'null');
insert into useractivity values
  (1, 1, '2005-01-02 13:00:00', '2005-01-02 16:00:00', '
     Discussed details');
insert into useractivity values
  (2, 2, '2005-02-03 11:00:00', '2005-02-03 14:00:00', '
     Discussion completed');
insert into useractivity values
  (3, 2, '2005-05-01 08:00:00', '2005-05-02 16:00:00', '50%
      accomplished');
insert into useractivity values
  (2, 1, current_timestamp, null, 'null');
```

```
insert into useractivity values
  (3, 1, current_timestamp, null, 'null');
insert into useractivity values
  (1, 2, current_timestamp, null, 'null');


insert into location values
  (0, 'null', 'null', 'null', 'null', 'null', 'null', 'null
     ', 'null', 'null', 'null');
insert into location values
  (1, 'Company Headquaters', 'description', '34.2348234', '
     16.3224234', 'Oxfordstreet 34', 'London', '', 'Great
     Britain', '+34 (4523) 898945', 'Jim Grandy');
insert into location values
  (2, 'Company Outpost', 'description', '14.789789', '
     13.879798', 'Downingstreet 1', 'New London', '', '
     Alaska', '+11 (8978) 234234', 'Jeanne Ho');


insert into userlocation values
  (0, 0, current_timestamp, null);
insert into userlocation values
  (1, 1, '2005-01-02 13:00:00', '2005-01-02 16:00:00');
insert into userlocation values
  (2, 1, '2005-02-03 11:00:00', '2005-02-03 14:00:00');
insert into userlocation values
  (3, 2, '2005-05-01 08:00:00', '2005-05-02 16:00:00');
insert into userlocation values
  (2, 2, current_timestamp, null);
insert into userlocation values
  (3, 1, current_timestamp, null);
insert into userlocation values
  (1, 1, current_timestamp, null);


insert into device(id, name, type, resolution, os,
    screenlayout, memory, storagecapacity, freestorage,
    calculatingcapacity, formats, preferreddeviceof, owner)
    values
  (0, 'null', 'null', 'null', 'null', 'null', 0, 0, 0, 0, '
     null', 0, 0);
insert into device(id, name, type, resolution, os,
    screenlayout, memory, storagecapacity, freestorage,
    calculatingcapacity, formats, preferreddeviceof, owner)
    values
  (1, 'workmate', 'laptop', '1024x768', 'Windows XP', '
     horizontal', 1024, 30000, 11923, 2130, 'pdf,doc,xls',
     1, 2);
insert into device(id, name, type, resolution, os,
    screenlayout, memory, storagecapacity, freestorage,
    calculatingcapacity, formats, preferreddeviceof, owner)
    values
  (2, 'dataslave', 'server', '800x600', 'Debian 3.1rev5', '
     horizontal', 4096, 450000, 293223, 4800, 'pdf,doc,xls,
     psd,tif,jpg', 2, 1);
insert into device(id, name, type, resolution, os,
    screenlayout, memory, storagecapacity, freestorage,
    calculatingcapacity, formats, preferreddeviceof, owner)
    values
  (3, 'digiti', 'pda', '320x240', 'Windows Mobile 2003 SE',
      'vertical', 64, 32, 17, 400, 'doc,xls,jpg', 3, 3);


insert into artifact (id, name, type, size, modified,
    status, owner, device) values
  (0, 'null', 'null', null, null, 'null', 0, 0);
insert into artifact (name, type, size, modified, status,
    owner, device) values
  ('report1.xls', 'Spreadsheet', 23422, current_timestamp,
     'opened', 1, 2);
insert into artifact (name, type, size, modified, status,
    owner, device) values
  ('whitepaper.doc', 'Microsoft Word Document', 151422,
     current_timestamp, 'locked', 2, 1);
insert into artifact (name, type, size, modified, status,
    owner, device) values
  ('specifications.pdf', 'Portable Document File', 54622,
     current_timestamp, 'closed', 2, 2);
insert into artifact (name, type, size, modified, status,
    owner, device) values
  ('HSqlDB.java', 'Java Source Code', 3422,
     current_timestamp, 'deprecated', 3, 3);


insert into userdevice values
  (0, 0, current_timestamp, null);
```

```
insert into userdevice values
  (1, 1, '2005-01-02 13:00:00', '2005-01-02 16:00:00');
insert into userdevice values
  (2, 3, '2005-02-03 11:00:00', '2005-02-03 14:00:00');
insert into userdevice values
  (3, 2, '2005-05-01 08:00:00', '2005-05-02 16:00:00');
insert into userdevice values
  (2, 1, current_timestamp, null);
insert into userdevice values
  (3, 2, current_timestamp, null);
insert into userdevice values
  (1, 3, current_timestamp, null);


insert into networkprofile(id, connectiontype, bandwidth,
    flatrate, wireless) values
  (0, 'null', 'null', 'false', 'false');
insert into networkprofile(id, connectiontype, bandwidth,
    flatrate, wireless) values
  (1, 'DSL', '2048x512', 'false', 'false');
insert into networkprofile(id, connectiontype, bandwidth,
    flatrate, wireless) values
  (2, 'Cable', '5120x1024', 'true', 'true');
insert into networkprofile(id, connectiontype, bandwidth,
    flatrate, wireless) values
  (3, 'GPRS', '86', 'false', 'true');


insert into devicenetworkprofile values
  (0, 0, current_timestamp, null);
insert into devicenetworkprofile values
  (2, 1, '2006-05-05 12:00:00', '2006-12-01 16:00:00');
insert into devicenetworkprofile values
  (1, 1, current_timestamp, null);
insert into devicenetworkprofile values
  (2, 2, current_timestamp, null);
insert into devicenetworkprofile values
  (3, 3, current_timestamp, null);
```

B.5 Example Server Configuration

Listing 10: Example server configuration

```
#-Dorg.osgi.framework.dir=\Storage Card\osgi\runtime-osgi/
```

```
    fwdir
#-Dorg.knopflerfish.verbosity=10
-Dlog4j.configuration=file:///Storage Card/Client1/log4j.
    properties
-Dat.tuwien.infosys.cfw.gps.commport=COM5
-Dorg.osgi.framework.system.packages=javax.sql,javax.
    microedition.io
-Dat.tuwien.infosys.cfw.debug=true
-Dorg.osgi.service.http.port=8080
-Dorg.knopflerfish.gosg.jars=file:/Storage Card/Client1/
    jars/*
#-Dorg.knopflerfish.framework.bundlestorage=file
#-Dorg.knopflerfish.framework.bundlestorage.file.reference=
    true
#-Dnet.slp.interfaces=127.0.0.1

-init
-initlevel 1
-install amigo_core.jar
-install cfw-metering-1.0.0.jar
-install http_all-2.0.0.jar
-install amigo_ksoap_binding.jar
-install jslp-osgi-0.99.2.jar
-install jsdk-2.2.jar
-install amigo_ksoap_export.jar
-install cm_all-2.0.0.jar
-install log_all-2.0.0.jar
-install servicebinder1.1.jar
-install log4j_bu.jar
-initlevel 2
-install cfw-context-store-api-1.0.0.jar
-initlevel 3
-install cfw-context-provider-1.0.0.jar
-install cfw-context-advertiser-1.0.0.jar
-initlevel 4
-install cfw-context-store-1.0.0.jar
-initlevel 5
-install cfw-context-consumer-1.0.0.jar
-initlevel 6
-install cfw-context-locator-1.0.0.jar
-initlevel 7
-install cfw-context-client-1.0.0.jar
-install cfw-demo-1.0.0.jar
```

```
-install cfw-gui-1.0.0.jar
-install cfw-random-ops-1.0.0.jar
-install cfw-threadpool-1.0.0.jar
-startlevel 10

-launch
-start amigo_core.jar
-start http_all-2.0.0.jar
-start amigo_ksoap_binding.jar
-start jslp-osgi-0.99.2.jar
-start amigo_ksoap_export.jar
-start cm_all-2.0.0.jar
-start log_all-2.0.0.jar
-start servicebinder1.1.jar
-start cfw-context-provider-1.0.0.jar
-start cfw-context-advertiser-1.0.0.jar
-start cfw-context-consumer-1.0.0.jar
-start cfw-context-locator-1.0.0.jar
-start cfw-gui-1.0.0.jar
```

C Technical information

C.1 Footprint

Bytes	Bundle name
660899	cfw-context-store-1.0.0.jar
325359	log4j_bu.jar
294741	framework.jar
136521	amigo_ksoap_binding.jar
99193	cfw-gui-1.0.0.jar
91766	http_all-2.0.0.jar
70173	servicebinder1.1.jar
66076	jslp-osgi-0.99.2.jar
62507	cm_all-2.0.0.jar
36063	log_all-2.0.0.jar
34555	jsdk-2.2.jar
28808	amigo_core.jar
23049	cfw-gps-1.0.0.jar
13893	cfw-context-store-api-1.0.0.jar
10143	cfw-threadpool-1.0.0.jar
9402	cfw-demo-1.0.0.jar
6412	cfw-context-client-1.0.0.jar
6232	cfw-random-ops-1.0.0.jar
6095	amigo_ksoap_export.jar
4952	cfw-context-provider-1.0.0.jar
4620	cfw-context-consumer-1.0.0.jar
4312	cfw-context-advertiser-1.0.0.jar
4222	cfw-context-locator-1.0.0.jar
2865	cfw-metering-1.0.0.jar

Table 3: Components of the framework packaged as Bundles

C.2 Package Structure

Bundle	imports	exports
context-store-api		(a.a.t.v.c)[1].contextstore (a.a.t.v.c).contextstore.entities
context-store	at.tuwien.infosys.cfw (a.a.t.v.c).entities	
context-locator	ch.ethz.iks.slp org.osgi.framework	(a.a.t.v.c).locator
context-advertiser	(a.a.t.v.c).contextstore ch.ethz.iks.slp org.osgi.framework	(a.a.t.v.c).advertiser
context-provider	(a.a.t.v.c).contextstore com.francetelecom.amigo.core org.osgi.framework org.ungoverned.gravity.servicebinder	(a.a.t.v.c).contextprovider
context-consumer	(a.a.t.v.c).contextstore (a.a.t.v.c).locator com.francetelecom.amigo.core org.osgi.framework	(a.a.t.v.c).contextconsumer
context-client	(a.a.t.v.c).contextconsumer (a.a.t.v.c).contextstore (a.a.t.v.c).contextstore.entities (a.a.t.v.c).locator org.osgi.framework	(a.a.t.v.c).metering
gps	org.osgi.framework (dynamic) javax.microedition.io	org.dinopolis.gpstool.gpsinput

Imports / Exports of the framework bundles
(a.a.t.v.c) = at.ac.tuwien.vitalab.cfw

C.3 Service imports and exports

Bundle	uses / registers	
context-store-api		
context-store	registers	(a.a.t.v.c).contextstore.ContextStore
	uses	(a.a.t.v.c).contextprovider.ContextStoreProvider
	uses	(a.a.t.v.c).advertiser.ContextStoreAdvertiser
context-locator	registers	(a.a.t.v.c).locator.ContextStoreLocator
	uses	ch.ethz.iks.slp.Locator
context-advertiser	registers	(a.a.t.v.c).advertiser.ContextStoreAdvertiser
	uses	(a.a.t.v.c).contextstore.ContextStore
	uses	ch.ethz.iks.slp.Advertiser
context-provider	registers	(a.a.t.v.c).contextprovider.ContextStoreProvider
	uses	(a.a.t.v.c).contextstore.ContextStore
	uses	com.francetelecom.amigo.core.AmigoServiceExporter
context-consumer	registers	(a.a.t.v.c).contextconsumer.ContextStoreConsumer
	uses	(a.a.t.v.c).contextstore.ContextStore
	uses	(a.a.t.v.c).locator.ContextStoreLocator
context-client	uses	(a.a.t.v.c).contextconsumer.ContextStoreConsumer
gps	registers	(a.a.t.v.c).gps.GPSService

Used / Registered OSGi services of the framework
(a.a.t.v.c) = at.ac.tuwien.vitalab.cfw

References

[1] AMIGO - Ambient intelligence for the networked home environment. http://www.hitech-projects.com/euprojects/amigo/index.htm. 03.08.2007.

[2] AMIGO Full Tutorial v2. http://amigo.gforge.inria.fr/obr/tutorial/amigo_full_tutorial_v2.zip. 03.08.2007.

[3] Concierge OSGi. http://flowsgi.inf.ethz.ch/concierge.html. 13.07.2007.

[4] Equinox OSGi. http://www.eclipse.org/equinox. 13.07.2007.

[5] Erste Schritte mit dem .NET Compact Framework. http://www.microsoft.com/germany/msdn/library/net/compactframework/ErsteSchritteMitDemNETCompactFramework.mspx. 11.07.2007.

[6] Gravity Service Binder. http://gravity.sourceforge.net/servicebinder/. 03.08.2007.

[7] hsqlDB - 100% pure Java Database. http://hsqldb.sourceforge.net/. 02.10.2007.

[8] hsqldb-cdc - A port of HSQLDB 1.8.0 from J2SE to the CDC platform. http://sourceforge.net/project/showfiles.php?group_id=102914\&package_id=157873. 27.07.2007.

[9] HyperText Transfer Protocol. http://www.w3.org/Protocols/. 25.09.2007.

[10] Introduction to WSDL. http://www.w3schools.com/wsdl/wsdl_intro.asp. 12.07.2007.

[11] JDBC Optional Package for CDC/Foundation Profile - Final Release (April 8, 2004). http://java.sun.com/products/jdbc/download.html\#cdcfp. 27.07.2007.

[12] JGrinder - A persistent Solution. http://jgrinder.sourceforge.net/. 03.12.2007.

[13] jSLP OSGi - SLP service discovery on OSGi platforms. http://jslp.sourceforge.net/jSLP-OSGi/index.html. 26.07.2007.

[14] JSR-000169 JDBC Optional Package for CDC/Foundation Profile. http://jcp.org/aboutJava/communityprocess/final/jsr169/. 02.10.2007.

[15] JSR-000172 J2ME^{TM}Web Services Specification. http://jcp.org/en/jsr/detail?id=172. 06.12.2007.

[16] Knopflerfish - Open Source OSGi. http://www.knopflerfish.org. 13.07.2007.

[17] Knopflerfish eclipse plugin. http://www.knopflerfish.org/eclipse_plugin.html. 24.01.2007.

[18] kSOAP 2. http://ksoap2.sourceforge.net. 30.07.2007.

[19] Mr Persister^{TM}- Complete Relational Persistence for Java. `http://www.jenkov.com/mrpersister/introduction.tmpl`. 02.10.2007.

[20] Oscar OSGi Framework. `http://forge.objectweb.org/projects/oscar`. 13.07.2007.

[21] ProSyst mBedded Server CLDC Edition. `http://http://www.prosyst.com/products/osgi_se_cldc_ed.html`. 13.07.2007.

[22] The Knopflerfish Axis port. `https://www.knopflerfish.org/svn/knopflerfish.org/trunk/osgi/bundles_opt/soap/axis.html`. 30.07.2007.

[23] Using J9W instead of J9 on Pocket PC devices. `http://www-1.ibm.com/support/docview.wss?uid=swg21209017`. 24.01.2007.

[24] W3C SOAP Specification. `http://www.w3.org/TR/soap`. 12.07.2007.

[25] Web Services Architecture Requirements. `http://www.w3.org/TR/2002/WD-wsa-reqs-20021011#IDAGWEBD`. 28.09.2007.

[26] WSIF - Web Services Invocation Framework. `http://ws.apache.org/wsif/`. 02.10.2007.

[27] . Biegel, V. Cahill. A framework for developing mobile, context-aware applications. In *Second IEEE Annual Conference on Pervasive Computing and Communications 2004*, pages 361–365. PerCom, 2004.

[28] A. K. Dey, G. D. Abowd. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer Verlag, London UK, 1999.

[29] A. K. Dey, G. D. Abowd, A. Wood. Cyberdesk: A framework for providing self-integrating context-aware services. In *Knowledge-Based Systems*, pages 3–13. 1999.

[30] OSGi Alliance. About the OSGi service platform. `http://www.osgi.org/documents/collateral/TechnicalWhitePaper2005osgi-sp-overview.pdf`. 12.02.2007.

[31] B. Schilit, M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network, 8(5)*, pages 22–32, 1994.

[32] B. Schilit, N. Adams, R. Want. Context-aware computing applications. In *1st International Workshop on Mobile Computing Systems and Applications*, pages 85–90. 1994.

[33] C. Dorn, D. Schall, S. Dustdar. Granular context in collaborative mobile environments. In *International Workshop on Context-Aware Mobile Systems (CAMS06), Oct. 31 - Nov. 1, Montpellier, France*. Springer Verlag, 2006.

[34] C.F. Sørensen, M. Wu, T. Sivaharan, G.S. Blair, P. Okanda, A. Friday, H. Duran-Limon. Context-aware middleware for applications in mobile ad hoc environments. In *ACM/IFIP/USENIX International Middleware conference 2ndWorkshop on Middleware*

for Pervasive and Ad-Hoc Computing (online proceedings). ACM/IFIP/USENIX, 2004.

[35] D. Schall, C. Dorn, S. Dustdar. Mobile Computing: Context-Aware. In *Encyclopedia of Wireless and Mobile Communications*. Taylor & Francis, 2008.

[36] D. Schall, M. Aiello, S.Dustdar. Web services on embedded devices. In *J. Web Infor. Syst. Vol.1, No. 1*. Troubador Publishing Ltd, 2005.

[37] D. Van Thanh, I. Jørstad, S. Dustdar. Mobile multimedia collaborative services. In Ismail Khalil Ibrahim, editor, *Handbook of Research on Mobile Multimedia*. Idea Group Publishing, USA, 2006.

[38] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[39] G. Alonso, F. Casati, H. Kuno, V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer Verlag, 2004. ISBN 3-540-44008-9.

[40] E. Guttman. SERVICE LOCATION PROTOCOL: Automatic discovery of IP network services. *IEEE Internet Computing*, pages 71–80, 0708 1999.

[41] Sun Microsystems Inc. CDC: Java Platform Technology for Connected Devices. `http://java.sun.com/products/cdc/wp/cdc-whitepaper.pdf`. 09.02.2007.

[42] Sun Microsystems Inc. CLDC HotSpot Implementation Virtual Machine. `http://java.sun.com/j2me/docs/pdf/CLDC-HI_whitepaper-February_2005.pdf`, 2005.

[43] J. Beatty, G. Kakivaya. Web Services Dynamic Discovery (WS-Discovery). `http://specs.xmlsoap.org/ws/2005/04/discovery/ws-discovery.pdf`, 2005.

[44] Li Gong. JXTA: A Network Programming Environment. *IEEE Internet Computing*, 2001.

[45] F. Rosenberg M. Baldauf, S. Dustdar. A Survey on Context Aware Systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2006.

[46] M. Divitini, B. A. Farshchian, H. Samset. UbiCollab: Collaboration support for mobile users. In *ACM Symposium on Applied Computing*, pages 1191–1195. ACM, 2004.

[47] Members of the UPnP Forum. Upnp device architecture 1.0. `http://www.upnp.org/resources/documents/CleanUPnPDA101-20031202s.pdf`, 2003.

[48] R. Muntz P. Castro. Managing Context Data for Smart Spaces. *IEEE Personal Communications, Vol.7, No.5*, October 2000.

[49] J. Pascoe. Adding generic contextual capabilities to wearable computers. In *2nd International Symposium on Wearable Computers*, pages 92–99. 1998.

[50] P.D. Costa, L.F. Pires, M. van Sinderen, J.P. Filho. Towards a service platform for mobile context-aware applications. In *1st*

International Workshop on Ubiquitous Computing, pages 48–61. IWUC, 2004.

[51] R. A. van Engelen, K. A. Gallivan. The gSOAP Toolkit for Web Services and Peer-To-Peer Computing Networks. In *2nd IEEE International Symposium on Cluster Computing and the Grid*. IEEE, 2002.

[52] T. Hofer, W. Schwinger, W. Retschnitzegger. Context-Awareness on Mobile Devices - the Hydrogen Approach. *IEEE Computer Society*, 2002.

[53] T. Strang. *Service Interoperability in Ubiquitous Computing Environments*. PhD thesis, L-M University Munich, 2003.

[54] T. Strang, C. Linnhoff-Popien. A Context Modeling Survey. In *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies table of contents, Grenoble, France*, pages 265–270. ACM Press, New York, NY, USA, 2005.

Index