Service Discovery in mobile networks by a combination of UDDI and LDAP

Dr. Kai-Oliver Detken
DECOIT GmbH
Fahrenheitstraße 1
D-28359 Bremen
+49-421-2208-185
detken@decoit.de

Patrick Mytanz
DECOIT GmbH
Fahrenheitstraße 1
D-28359 Bremen
+49-421-2208-187
mytanz@decoit.de

ABSTRACT

The ever increasing number of nomadic users will create a trend towards networking technologies that support mobility. Although wireless networks have already become commonplace and continuously expand their area of application with new technologies, like wireless LAN (WLAN), wire-line networks as backbones or broadband bearer technologies. A scenarios where the user has the requirement of using all available services on the road just as he would at home, in other words without needing to know any details about the network infrastructure – whether it will be by plugging the laptop to hotel network or turning on WLAN on the PDA – will become more and more common. It is in situations like these you need automatic discovery mechanisms to get the network connection and find the available services on this network. Therefore we need a service discovery which works independent from the underlying infrastructure. This paper describes a solution to get such service discovery for nomadic users through a mixture of existing protocols and standards with UDDI, SOAP, LDAP, and SLP.

Categories and Subject Descriptors

D.2.7 [**Distribution, Maintenance, and Enhancement**]: *Enhancement, Extensibility, Portability and Restructuring of standard protocols in the area of service discovery.*

General Terms

Performance, software design, reliability, experimentation & tests, standardisation, verification.

Keywords

mobile networks, service discovery, UDDI, LDAP, SOAP, SLP

1. INTRODUCTION

The term "Web service" describes specific business functionality exposed by a company, usually through an Internet connection, for the purpose of providing a way for another user entity (i.e. another company, software or private user) to use the service. Web services as an W3C (WWW Consortium) activity aim to create an industry standard for common programmatic interfaces enabling transparent application to application or machine to machine (M2M) communication on the World Wide Web (WWW).

Web services are an attempt at making programmable application logic accessible using standard Internet protocols. Web services in this sense combine component-based development and Web technologies. Like components, Web services represent black-box functionality that can be reused without caring how they are implemented. Unlike current component models, Web services are not accessed via object-model-specific protocols, such as the Distributed Component Object Model (DCOM), Remote Method Invocation (RMI), or Internet Inter-ORB Protocol (IIOP). Instead, Web services are accessed via ubiquitous Web protocols and data formats, such as Hypertext Transfer Protocol (HTTP) and Extensible Markup Language (XML), which could be simply referred to Simple Object Access Protocol (SOAP). Furthermore, Web Services Description Language (WSDL) is introduced to define and describe the interface of SOAP messages. Web services can be implemented on any platform in any programming language, as long as they can create and consume the messages defined for the Web service interface. To enable a global services market, Universal Discovery Description and Integration (UDDI) is set to play the role of centralised registry of services. In short, to release the flexibility and extensibility from proprietary to public when building the Internet application, Web services is loosely coupled rather than tightly coupled for realising the reuse vision of component.

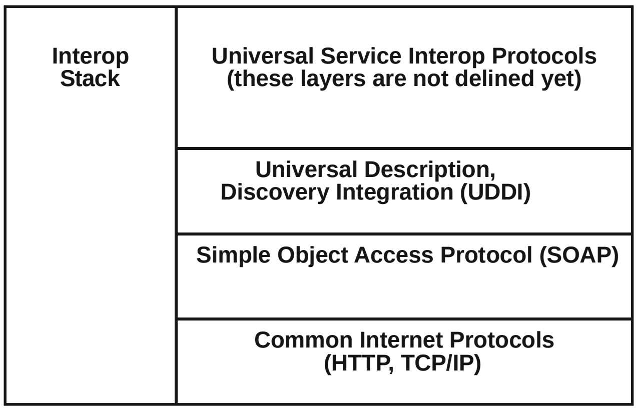
Interop Stack
Universal Service Interop Protocols (these layers are not delined yet)
Universal Description, Discovery Integration (UDDI)
Simple Object Access Protocol (SOAP)
Common Internet Protocols (HTTP, TCP/IP)

Figure 1. Web services protocols.

Additionally, mobile access of such Web services offer new possibilities to interact between user and network related new contexts and information. For example, it could be interesting from which place the user wants to use available services and also the user don't know what kind of service is available. This is the reason why we need an automatic and transparent service discovery. This paper shows actual standards and protocols and an own solution to get a service discovery platform.

2. SERVICE MANAGEMENT REQUIREMENTS

This section gives a short overview of currently available service management solutions and related protocols with a special focus on service discovery:

- Service definition not only includes what the service should do but also who is entitled to use it (security).
- Creation and deployment are the means by which a service is produced and put to work.
- Announcement and discovery describes the techniques employed to inform users of the existence and availability of a service.

Service management describes the various ways a service or resource can be defined, created, deployed, announced, discovered, delivered and consumed.

The following relevant protocols and standards have to take into account regarding the development of a service discovery approach:

1. Service Location Protocol (SLP)
2. Lightweight Directory Access Protocol (LDAP)
3. Universal Description, Discovery and Integration (UDDI)
4. Web Service Description Language (WSDL)
5. Simple Object Access Protocol (SOAP)

The Service Location Protocol (SLP) is an IETF standard track protocol that provides a framework to allow networking applications to discover the existence, location, and configuration of networked services in enterprise networks. Traditionally, in order to locate services on the network, users of network applications have been required to supply the host name or network address of the machine that provides a desired service. Ensuring that users and applications are supplied with the correct information has, in many cases, become an administrative nightmare. Protocols that support service location are often taken for granted, mostly because they are already included in many network operating systems. Additionally, an IETF service location protocol was not standardised until the advent of SLP. Because it is not tied to a proprietary technology, SLP provides a service location solution that could become extremely important. [6]

The second protocol which is important for service discovery is LDAP (RFC2251 –Lightweight Directory Access Protocol). LDAP is a directory access protocol based on the X.500 schema, the original design for a directory service designed by the ISO as part of OSI. X.500 was in this sense designed to support a world-wide address-book. LDAP is a lightweight version of this technology designed by the IETF to run over TCP/IP instead of OSI. There are many different ways to provide a directory service. Different methods allow different kinds of information to be stored in the directory, place different requirements on how that information can be referenced, queried and updated, how it is protected from unauthorised access, etc. Some directory services are local, providing service to a restricted context (e.g., the finger service on a single machine). Other services are global, providing service to a much broader context (e.g., the entire Internet). Global services are usually distributed, meaning that the data they contain is spread across many machines, all of which co-operate to provide the directory service. Typically a global service defines a uniform namespace which gives the same view of the data no matter where you are in relation to the data itself. The Internet Domain Name System is a further example of a globally distributed directory service.

UDDI stands for Universal Description, Discovery and Integration. The UDDI Project is an industry initiative that is working to enable businesses to quickly, easily, and dynamically find and transact with one another. UDDI provides functionality necessary in order to describe services, discover desired services, and integrate services into service packages. The UDDI specification is based on World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF) standards such as Extensible Markup Language (XML), HTTP, and Domain Name System (DNS) protocols. Additionally, the UDDI specification addresses cross platform-programming features by adopting the SOAP messaging specification. The UDDI specification does not dictate service registry implementation details. The UDDI specification defines an XML-based data model and a set of SOAP APIs to access and manipulate that data model. The SOAP APIs define the behaviour a UDDI registry must exhibit.

As communications protocols and message formats are standardised in the Web community, it becomes increasingly possible and important to be able to describe the communications in some structured way. WSDL addresses this need by defining an XML grammar for describing network services as collections of communication endpoints capable of exchanging messages. WSDL service definitions provide documentation for distributed systems and serve as a recipe for automating the details involved in applications communication. [6]

The Simple Object Access Protocol (SOAP) is a protocol that operates over HTTP implementing remote procedure calls in XML. This approach allows for better interoperability, standards conformance and simplicity as it allows for the usage of any platform or programming language and all messages exchanged are in a human readable format, which is also quite easy to understand. Both the body of the request (procedure call) when a procedure executes on the server and the value it returns are always formatted in XML. Parameters and returned values can be simple as well as complex data types.

The service discovery middleware software, which has been developed, had to provide flexible workflow set-up for “composite” services. The user interface had to be simple, configurable and reflect the required parameters corresponding to a service. Users want to control whether advertisements or other kind of information are “pushed” on their terminals. They also prefer to impose their own criteria while locating services. The ability to locate the position of a mobile device is a key to providing geographically specific value-added information. Using available positioning techniques like GPS, the client software provides location information to the service discovery middleware. This software middleware has been developed within the European project NOMAD (Integrated Networks for Seamless and Transparent Service Discovery). In this paper we focused only to the service discovery aspect and the realisation of this approach.

2.1 UDDI and SOAP

Universal Description, Discovery and Integration (UDDI) specifically focuses on functionality for the deployment and discovery of Web services. The concept behind UDDI is based on a distributed registry of service descriptions described with the help of a common XML format. A common analogy used for UDDI is a "phone book for Web services." It contains names, mailing addresses, contacts, contact phone numbers, Web services offered, addresses of Web services (i.e. URLs), meta-data describing the "interfaces" of Web services, etc.

Using the UDDI discovery services, providers of a certain service or product individually register information about the Web services that they expose for use by others. This information can be added to the UDDI registry via tools that make use of the UDDI API. The UDDI registry is a logically centralised, physically distributed database with multiple root nodes that replicate data with each other on a regular basis. Once a user registers with a single node of the registry, the data is automatically shared with other UDDI root nodes and becomes freely available to anyone who needs to discover what Web services are available.

Interactive applications or software registering itself automatically can access the registry over the Internet using the UDDI API can do this either based on SOAP/XML messages or one of the already freely available client-side Java, Visual Basic, C#, or COM API's for accessing UDDI registries. Currently operational UDDI repositories include freely available facilities from HP, IBM, SAP, and Microsoft.

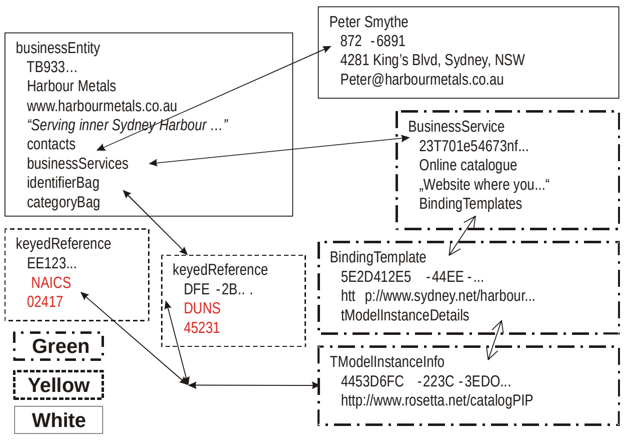

Figure 2. UDDI Registration example.

The core components of a UDDI registry are Business Registrations and Service Type Registrations. These registrations contain XML data used to describe a business entity and its Web services. Conceptually, the information provided in a UDDI business registration consists of three components:

1. *white pages* including address, contact, and known identifiers of the organisation registered
2. *yellow pages* including industrial categorisations based on standard taxonomies
3. *green pages*, the technical information about services that are exposed by the business. Green pages include references to specifications for Web services, as well as support for pointers to various file and URL based discovery mechanisms if required.

Service Type Registrations on the other hand fulfil the following functions:

1. Provide a pointer to the namespace where service type is described, in order to help the user or application understand how to use the service
2. Provide "service generator" identifiers
3. Provide identifiers for the service type registration, used as a signature by web sites that implement those services

The UDDI distributed service registry is a distributed database modelled after the Internet DNS (Domain Name Service). As such it is composed of nodes that carry a replicated version of the UDDI.org central service repository. Replication is executed on a daily basis. This allows for a complete set of registered services on all the nodes of the UDDI distributed databases with a maximum delay of one day. Registration of new services can be accomplished either directly to the UDDI registry or via peer nodes that could be front-ends (possibly in the form of WWW pages) operated by ASPs. Such peer nodes would internally also make use of the common SOAP API supported by all nodes and the central repository. Compliance to the rules of UDDI.org and compatibility are enforced by appropriate contracts.

In a typical scenario, the UDDI registry is expected to be accessed via SOAP over HTTP again over a conventional Internet connection. This means that when accessing the registry the user will mainly communicate with the Web server representing his chosen UDDI registry node in the Internet. Messages received by the Web server are then processed with the help of XML/SOAP processors that transform them into queries and other appropriate UDDI function calls. These calls are then executed on the local UDDI registry service node. The actual service data does not have to be stored within the node, but can again be queried from a differed backend. Finally the transaction if concluded by transferring the requested data back to the user through the Web-Server over HTTP.

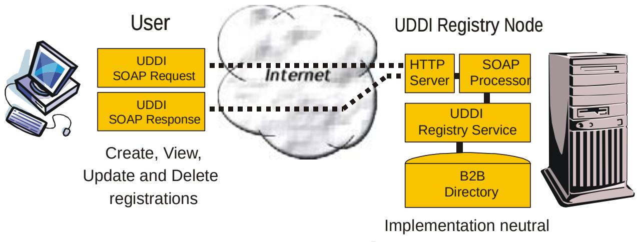

Figure 3. UDDI and SOAP.

2.2 SLP architecture

As computers become more portable and networks larger and more pervasive, the need to automate the location and client configuration for network services also increases. The Service Location Protocol (SLP) is an IETF standard being developed by the IETF that provides a scalable framework for networking applications to discover the existence, location, and configuration of networked services in enterprise IP networks. SLP is currently available in Version 2. [6]

The SLP architecture consists of three main components:

1. *User Agents (UA)* perform service discovery, on behalf of the client (user or application)
2. *Service Agents (SA)* advertise the location and characteristics of services, on behalf of services

3. *Directory Agents (DA)* collect service addresses and information received from SAs in their database and respond to service requests from UAs

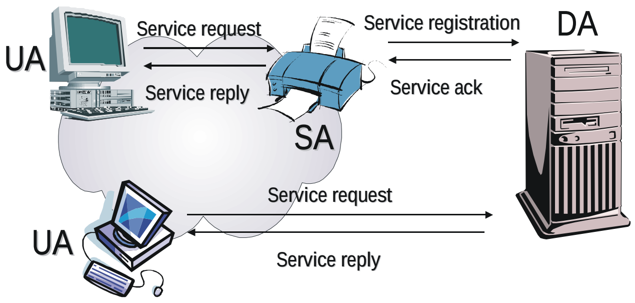

Figure 4. SLP Architecture.

The previous figure shows the interactions between the three agents. When a new service connects to a network, the SA contacts the DA to advertise its existence (Service Registration). When a user needs a certain service, the UA queries the available services in the network from the DA. After receiving the address and characteristics of the desired service, the user may finally utilise the service. Before a client (UA or SA) is able to contact the DA, it must discover its existence. There are three different methods for DA discovery:

1. **Static:** With static discovery, SLP agents obtain the address of the DA through DHCP (Dynamic Host Configuration Protocol). The necessary DHCP options for SLP are defined in RFC-2610. DHCP servers then distribute the addresses of DAs to hosts that request them.

2. **Active:** In active discovery, UAs and SAs send service requests to the SLP multicast group address (239.255.255.253). A DA listening on this address will eventually receive a service request and respond directly (via unicast) to the requesting agent.

3. **Passive:** In case of passive discovery, DAs periodically send out multicast advertisements for their services. UAs and SAs learn the DA address from the received advertisements and are now able to contact the DA themselves via unicast

It should be emphasised that the DA is not mandatory. DAs should be deployed in large networks with many services, as they implicitly help categorise services into different groups (scopes). In smaller networks (e.g., home or car networks) it is more effective to deploy SLP without a DA. SLP has therefore two operational modes, depending on whether a DA is present or not. If a DA exists on the network, it collects all service information advertised by SAs thus automatically playing the role of a local service repository. UAs will send their service requests to the DA and receive the desired service information. If there is no DA, UAs repeatedly send out their service request to the SLP multicast address. All SAs listen for these multicast requests and, if they advertise the requested service, they will send unicast responses to the UA. Furthermore SAs broadcast the existence of new services periodica1ly by mu1ticasting announcements of their existence. This mechanism makes sure that UAs are kept informed. Services are advertised using a service URL and a service template. The service URL contains the IP address of the service, the port number, and path. Service templates specify the attributes that characterise the service and their default values. [7]

2.3 LDAP model

The LDAP information model is based on entries. An entry is a collection of attributes that has a globally-unique Distinguished Name (DN). The DN is used to refer to the entry unambiguously. Each of the entry's attributes has a type and one or more values. The types are typically mnemonic strings, like “cn” for common name, or “mail” for an e-mail address. The syntax of values depend on the attribute type is.

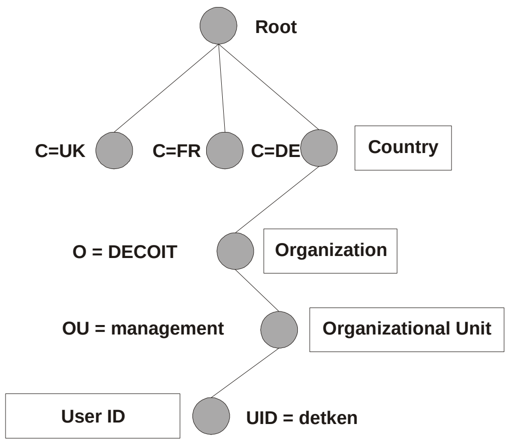

Figure 5. LDAP Directory Information Tree (DIT).

In LDAP, directory entries are arranged in a hierarchical tree-like structure. Traditionally, this structure reflected the geographic and/or organisational boundaries. Entries representing countries appeared at the top of the tree. Below them are entries representing states and national organisations. Below them might be entries representing organisational units, people, printers, documents, or just about anything else you can think of. Figure 5 shows an example LDAP directory tree using traditional naming.

The tree may also be arranged based upon Internet domain names. This naming approach is becoming increasing popular as it allows for directory services to be locating using the Domain Name System (DNS).

An entry is referenced by its distinguished name, which is constructed by taking the name of the entry itself (called RDN – Relative Distinguished Name) and concatenating the names of its ancestor entries. The full DN format is described in RFC-2253. In addition, LDAP allows you to control which attributes are required and allowed in an entry through the use of a special attribute called object class. The values of the object class attribute determine the schema rules the entry must obey.

LDAP defines operations for querying and updating the directory. Operations are provided for adding and deleting an entry from the directory, changing an existing entry, and changing the name of an entry. Most of the time, though, LDAP is used to search for information in the directory. The LDAP search operation allows some portion of the directory to be searched for entries that match some criteria specified by a search filter. Information can be requested from each entry that matches the criteria.

One or more LDAP servers contain the data making up the LDAP directory tree. An LDAP client connects to an LDAP server and asks it a question. The server responds with the answer and/or with a pointer to where the client can get additional information (typically, another LDAP server). This mechanism is known as an LDAP referral. When executing an active Referral the server obtains the required information from other systems and delivers it to the client, in passive Referrals the server simply returns a Reference.

This mechanism allows a client to effectively see the same view of the directory no matter which LDAP server a client connects to. Or in different terms, a name presented to one LDAP server, references the same entry it would at another LDAP server. This is an important feature of a global directory service, like LDAP.

3. SERVICE DISCOVERY PLATFORM

This service discovery architecture is based on the concept of integrating a distributed service repository based on a hierarchical infrastructure and a technology for ad-hoc service discovery in order to allow for both efficiency and scalability, but at the same time also ad-hoc operation and flexible update functionality.

3.1 Architecture

In order to provide efficient, user-centric Service Discovery services such as those defined we needs to address the following issues:

1. Efficiency
2. Scalability
3. Ad-Hoc operation, i.e. without prior configuration
4. Dynamic service update support

Existing technologies focus on a subset of the aforementioned issues thereby offering different combinations of advantages and disadvantages:

1. Centralised repositories are efficient but inherently not scalable
2. Replicated databases are more scalable than centralised solutions, but not truly distributed and additionally suffer from update and synchronisation issues
3. Architectures based on existing infrastructure are efficient, but require additional configuration every time the infrastructure (or the locality) changes
4. Ad-Hoc networks are very adaptive and easy to set-up, but have high overhead and are not scalable

This service discovery architecture is based on the concept of integrating a distributed service repository based on a hierarchical infrastructure and a technology for ad-hoc service discovery in order to allow for both efficiency and scalability, but at the same time also ad-hoc operation and flexible update functionality (see Figure 6). In accordance with the concept of integration and extension of COTS (Commercial-Off-The-Shelf) technologies a directory service based on LDAP was chosen as a distributed database and hierarchical service repository. [7]

Employing a distributed hierarchical infrastructure (see Figure 7) minimises signalling traffic and allows for efficient transport of queries resulting during the service discovery process, thereby improving scalability. The need for optimisation in this respect results from the requirement of operation over the Internet. The distributed nature of such a technology also allows for updates of service related data directly in the branch of the tree (and the database) that is responsible for the service without any conflicts with other nodes or parts of the database.

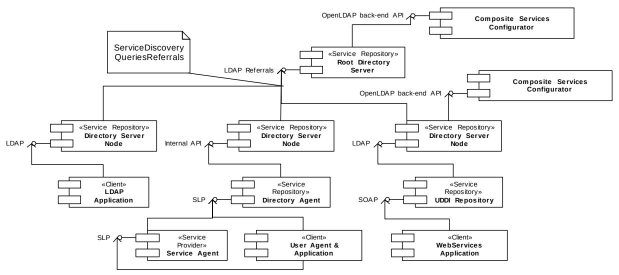
OpenLDAP back-end API
Composite Services Configurator
ServiceDiscovery QueriesReferrals
«Service Repository» Root Directory Server
LDAP Referrals
Composite Services Configurator
OpenLDAP back-end API
«Service Repository» Directory Server Node
LDAP
«Service Repository» Directory Server Node
Internal API
«Service Repository» Directory Server Node
LDAP
«Client» LDAP Application
«Service Repository» Directory Agent
SLP
«Service Repository» UDDI Repository
SOAP
SLP
«Service Provider» Service Agent
«Client» User Agent & Application
«Client» WebServices Application

Figure 6. Service Discovery platform components.

Furthermore, integration of a technology for ad-hoc service discovery allows for "Plug'n Play" discovery of the necessary hierarchy nodes as well as operation in environments where Internet connectivity is not available (see Figure 8).

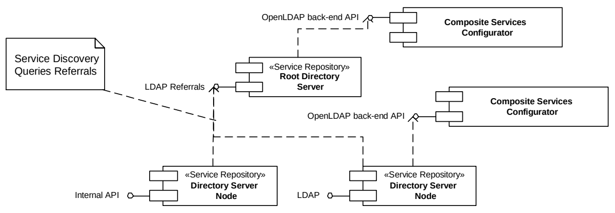
OpenLDAP back-end API
Composite Services Configurator
Service Discovery Queries Referrals
«Service Repository» Root Directory Server
LDAP Referrals
Composite Services Configurator
OpenLDAP back-end API
«Service Repository» Directory Server Node
Internal API
«Service Repository» Directory Server Node
LDAP

Figure 7. Distributed Service Repository.

The provision of functionality for on-the fly creation of composite services requires the integration of a composite service configuration engine. This component will be specified based on object oriented variant configuration concepts and integrated into the LDAP Directory Information Tree (DIT) as an LDAP node, using the OpenLDAP back-end API. This work involves the adaptation of configuration concepts to fit the hierarchical structure of a distributed structure and query functionality of directory service. Moreover a new product data model for composite services has been specified on the basis of an LDAP Schema.

Current implementations of LDAP offer flexible database integration mechanisms that make the coupling of a large variety of systems possible through a simple and well documented interface. Servers like the open source product OpenLDAP are based on a modular front-end-backend architecture that allows that usage of the LDAP front-end with arbitrary back-ends. Such back-ends would traditionally be relational or other databases (SQL, BDB, LDBM, etc.), programmable back-ends (i.e. perl, tcl, etc.) or even other LDAP servers, LDAP proxies and other constructs. OpenLDAP can be configured to serve multiple back-ends at the same time. This means that a single OpenLDAP server can respond to requests for many logically different portions of the LDAP tree, using the same or different back-ends for each part of the DIT.

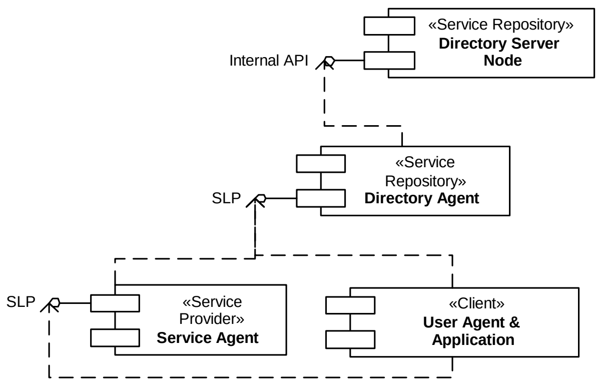

Figure 8. LDAP Directory Server & SLP Directory Agent integration

Leveraging current development in the area of Web services by providing access to the distributed service repository and service discovery functionality as well as the composite service configuration engine requires the specification of a UDDI/SOAP interface for the service discovery middleware (see Figure 9). Furthermore a mapping between the UDDI service schema and other schemata employed had to be provided. [6]

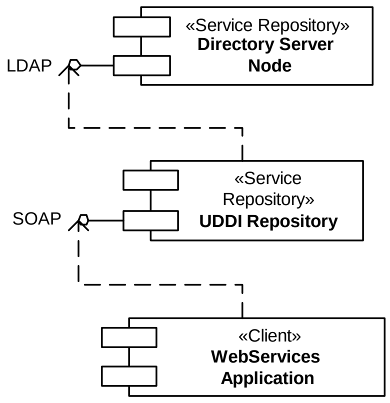

Figure 9. SOAP/UDDI access to the LDAP service repository.

3.2 Services composition based on LDAP

In accordance with the concept of integration and extension of COTS technologies a directory service based on the Lightweight Directory Access Protocol (LDAP) was chosen as a distributed database and hierarchical service repository.

In LDAP, directory entries are arranged in a hierarchical tree-like structure called the Directory Information Tree (DIT). Traditionally, this structure reflected the geographic and/or organisational boundaries. Entries representing countries appeared at the top of the tree. Below them are entries representing states and national organizations. Below them might be entries representing organisational units, people, printers, documents, or just about anything else you can think of. The DN format used by LDAP is defined in RFC-2253.

Current implementations of LDAP offer flexible database integration mechanisms that make the coupling of a large variety of systems possible through a simple and well documented interface. Servers like the OpenSource product OpenLDAP- are based on a modular front-end-backend architecture that allows that usage of the LDAP front-end with arbitrary back-ends. Such back-ends would traditionally be relational or other databases (SQL, BDB, LDBM, etc.), programmable back-ends (i.e. perl, tcl, etc.) or even other LDAP servers, LDAP proxies and other constructs. OpenLDAP can be configured to serve multiple back-ends at the same time. This means that a single OpenLDAP server can respond to requests for many logically different portions of the LDAP tree, using the same or different back-ends for each part of the DIT.

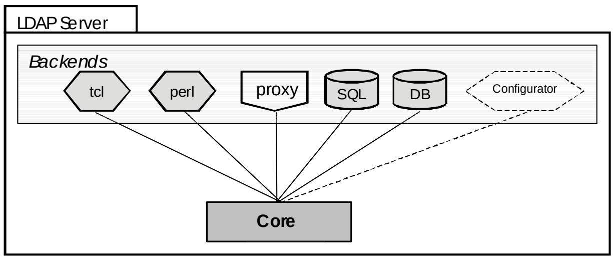

Figure 10. OpenLDAP frontend-backend architecture.

Middleware for the management and configuration of composite services can be thus integrated as an additional backend. This backend is responsible for resolving queries regarding composite services based on defined constraints. The configurator itself could also use the LDAP distributed database as a source for data on elementary services. Such queries would then be referred to the appropriate LDAP-Node within the DIT. The configurator itself can also work locally as supporting module for the local service discovery node or be installed centrally.

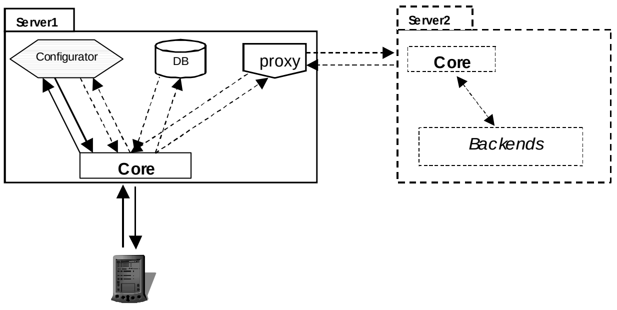

Figure 11. Service composition middleware as LDAP backend.

The OpenLDAP back-end API can be used to implement components that can be accessed by an OpenLDAP directory node as a database back-end. The composite service configuration engine can be thus seamlessly integrated into the distributed directory hierarchy by implementing an interface based on the aforementioned API, thereby transforming it into a database back-end component for an OpenLDAP directory node.

Each Back-end can be compiled as a static or a dynamic module, depending which way it is going to be needed. It is quite easy to add new back-ends, the configurator module is implemented as such a backend. [7]

3.3 Configurator back-end implementation

Composite service configurator can be configured to receive service discovery queries directed to the LDAP DIT node they are attached to. This is achieved by designating the configurator back-end as responsible for a predetermined branch of the DIT. The configurator being one of many possible back-ends of a single LDAP server can be restricted to a portion of the branch covered by the directory server itself. This allows standard-conform access to the configurator via LDAP queries. At the same time the configurator has full access to all the LDAP DIT and can itself use LDAP queries to gather information on additional services, as well as access other configurators located in different parts of the distributed directory. Such an arrangement allows recursive creation of composite services, where a configurator can consult any number of other Engines providing some subset of the overall composite service.

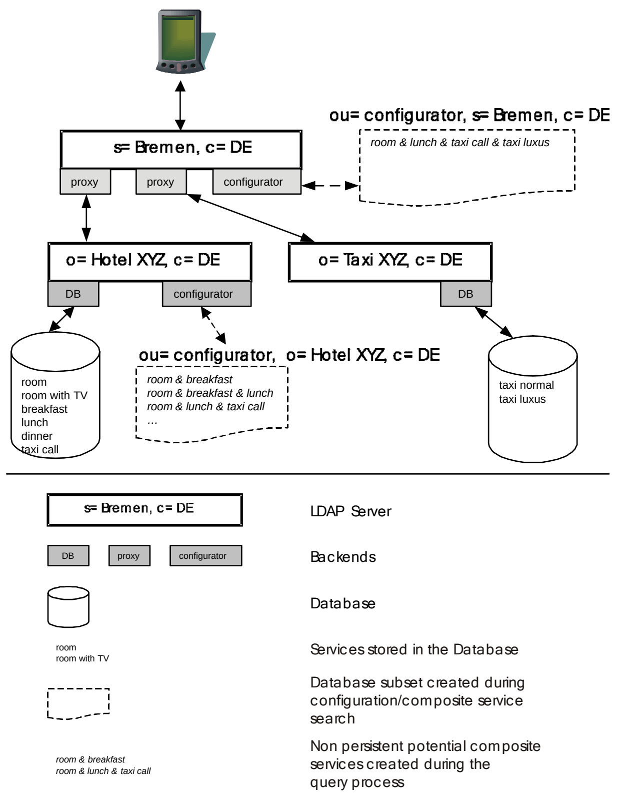

Figure 12. Distributed directory hierarchy with multiple configurator engines.

The configurator handling composite services for a service repository in the area of Bremen could for instance be set-up as one of the back-ends for a directory server with the distinguished name (DN) “l=Bremen, c=DE”. Queries related to elementary services under the aforementioned DN are automatically handled by the root directory server or forwarded to other associated directory servers handling smaller branches of the local DIT (i.e. Hotel XYZ, Taxi XYZ, etc.). Queries related to composite services on the other hand are addressed to a specific branch of the DIT (ou=configurator, l=Bremen, c=DE) and are forwarded to the configurator back-end.

The configurator has been implemented in ANSI C, making it platform independent. The basic idea of the configurator is to hold a virtual tree for each active connection (session) to it, that only exists as long as the connection exists. Such a connection is anonymous; for this reason each connection has a separate user (object) corresponding to it. The connections are each independent from the others, so no user from one connection has access to data from other connections. [7]

3.4 Using composite services in LDAP

In the LDAP API are a lot of functions, which make it possible to manage services in different manners: reading, writing and updating are just a few examples that were shown in the previous chapters. If we use a composite service instead of a common service, the existing functions are not adequate anymore, so a new way to handle these more complex services has to be found. The LDAP API itself shall not be changed; instead we define a way to solve even those composite services with only the available functions For this matter, there are three possible scenarios that have to be considered, where the first scenario is the common scenario that is currently possible with the LDAP API:

1. Elementary service
   a. User selects a service type
   b. User defines the search criteria
   c. User makes a query
   d. User selects the service
2. Using composite services
   a. User selects predefined composite service (as a filter maybe stored in the client)
   b. User defines search criteria for the individual service types
   c. User performs query for available services
   d. User selects a specific service for each service type
   e. User defines connection data for services with multiple interfaces
   f. Each time a new query is made, the user has to confirm choice of the compatible services
3. Editing/creating composite services
   a. User selects service types out of a list
   b. Composite service object as a higher node in the tree
   c. Composite service components contain seq_num, a filter describing useful services and a description
   d. User chooses the type of workflow (type of workflow is implicitly stored in the seq_num)
   e. Define relationships according to the workflow

3.5 The LDAP backend

Web services are services that are made available from a business Web server for distant Web users or other Web-connected programs. Providers of Web services are generally known as application service providers (ASP). Besides the standardisation and wide availability to users and businesses of the Internet itself, Web services are also increasingly enabled by the use of the extensible markup language (XML) as a means of standardising data formats and exchanging data. XML is the foundation for the web services description language (WSDL).

To develop the LDAP backend as a new backend, the LDAP Class Library for Java (JLDAP) includes the most important methods to access, manage, update, and search for information stored in directories accessible using LDAPv3. The main class is the LDAP connection, which contains methods to connect to the LDAP back-end, and add or delete an entry.

SOAPUDDI is a Java-based software, which is emulating a UDDI Server in its base functionality, being conform to the UDDI 2.0 specifications (UDDI = Universal Description, Discovery, and Integration). SOAPUDDI acts like a normal UDDI Server using Tomcat in version 4.1.24 as the servlet engine. A Linux server has been successfully configured with this software.

SOAPUDDI is using a relational database to store the necessary information. Regarding scalability and performance it is more efficient to use a directory backend to store this information. Since this kind of information will be stored once and read several times, the reading-optimised LDAP is the faster choice (performance). Further, such directory-backend can distribute data over several databases of different size and content (scalability). The SOAPUDDI API had to be modified, including minor modifications in roundabout 50 classes regarding backend connectivity.

SOAPUDDI has two sections for publishing and querying LDAP directories with major differences between each other: The first section is called "Publish Section"', the second is the "Inquiry section". The methods in these two sections were re-written, so that they won't access the UDDI back-end (in this case a normal database), but instead use JLDAP to access the LDAP directory. One of the main advantages is that an LDAP directory can be distributed over several computers and so databases, and is accessible from everywhere in the Internet.

SOAPUDDI offers a list of Web services with their corresponding URL to the clients, including additional information like description, parameters etc. To manage Web services in the LDAP directory, the first step is to collect them. For this purpose, a normal search for Web services is accomplished in the "Inquiry Section". All necessary information is gained from the SOAP messages received from the SOAP-client. For example, if the user is searching for a specific name, Afterwards, the read LDAP entities have to be converted into the UDDI format.

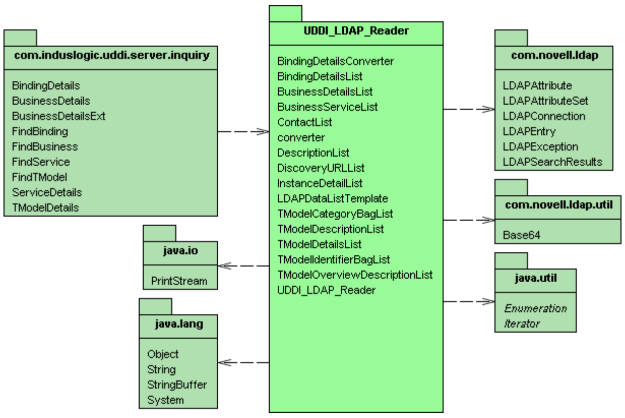

Figure 13. UDDI_LDAP_Reader package.

The LDAPConnection entity represents a direct binding to the LDAP directory and provides methods to search, update or delete LDAP entries. The result of a successful search is represented by a LDAPSearchResult.

To publish the new Web services in the "publish Section" we have to convert them to create an LDAP object. For this we have to read the received data from the SOAP message and decide which task we have to do.

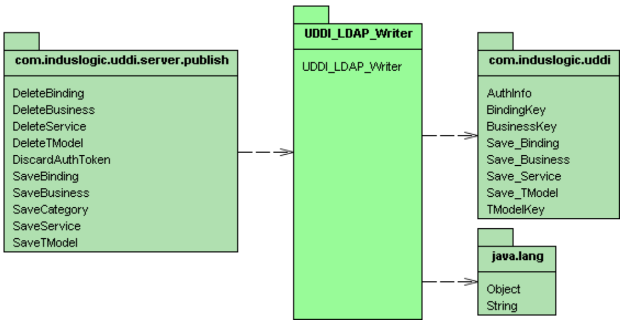

Figure 14. UDDI_LDAP_Writer package.

To publish the BusinessDetails, the UDDI_LDAP_Writer is used. This class is generating or modifying the LDAPEntries, based on the read UDDI data. The actual transforming will be done by the LDAPEntryWriter. This object is transforming the UDDIObjects to LDAPEntry. The main method is the write() method which accepts an UDDIObject and a String for the case of transforming this object. A case for transforming can be "Contact". This method always returns the new UDDI business key of the transformed and written UDDIObject.

The following diagram shows the way of communication between the UDDI browser and JLDAP. The UDDI browser uses UDDI4J to connect to a modified SOAPUDDI software and is acting as a usual UDDI server. This software is called UDDI_LDAP_CONVERTER. The backend of this converter uses JLDAP to read and write a normal LDAPv3 conform directory.

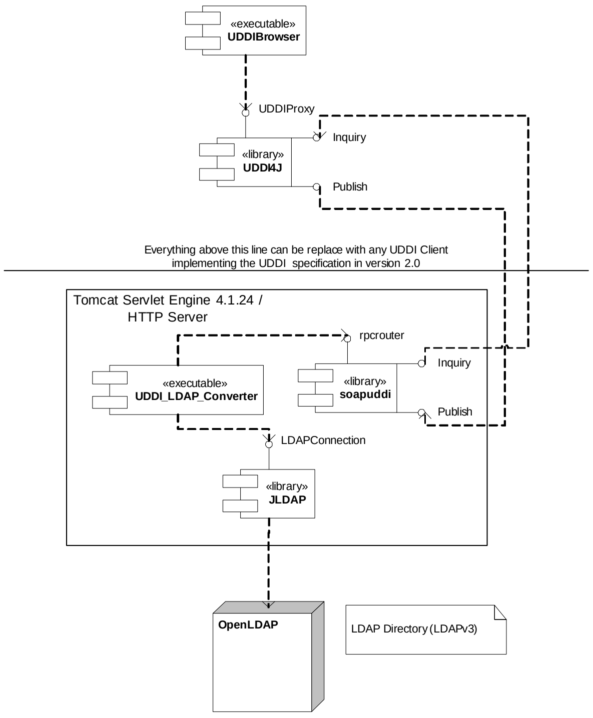

Figure 15. Software components overview

To be independent from platform and operating system the software has been implemented in Java. In addition this has the benefit that various existing software libraries in Java are available and can be used. However this fact will strongly affect the development process and reduce the durability of the development. For the subsequence software, a Web server with integrated servlet engine is needed. Here Apache's Tomcat in the Linux version has been figured out as the best solution.

The Java development environment is JBuilder in version 9. This is free of charge available for research purposes and for private use. JBuilder 9 had in addition integrated CVS functionalities. For the CVS repository the development server is used. This serves version control and offers besides backup functionality for all project data. [7]

Table 1. Software development components

Components	Software
Programming language	Java(TM) 2 SDK, Standard Edition Version 1.4.1
Server	Linux Debian Woody, Testing Release 3.0
Web-Server / Servlet Engine	Tomcat 4.1.24 for Linux
Development environment	JBuilder 9 Personnel for Windows
Version control system	CVS (Concurrent Versioning System)

4. CONCLUSIONS

Service discovery is an efficient way to recognise a service in a foreign network automatically. The first approach is that we use a central database to find available services. Each service was assigned to a category, which offers information about the service to the user. The second decentralise approach includes different logical areas on several server systems. This requires a continuously communication between all components. After a test phase one result was that the centralise data approach is the only possibility for mobile users to get the services just in time with the correct information. The scalability problem in such model has been solved by a central repository which gets its information from different databases. That is possible by the use of LDAP and UDDI as the most relevant protocol standards which exist for service discovery.

This paper has outlined the global architecture of the approach, divided into different modules and their interdependencies. In subsequent chapters, the details of LDAP based service composition backend, the LDAP/SLP integration backend as well as the Web service LDAP-UDDI backend have been described.

The European project NOMAD developed a service discovery platform which is scalable and efficient. Additionally this solution don't need any prior configuration and make a dynamic service update possible, automatically. Actually, this prototype of a service discovery middleware described in this paper will be tested, evaluated and further refined.

5. ACKNOWLEDGMENTS

The project NOMAD (IST-2001-33292) is funded by the European Commission as IST project. The authors wish to acknowledge the European Commission for their support. The authors also wish to acknowledge their gratitude and appreciation to all NOMAD partners for their strong support and valuable contribution during the various activities presented in this paper.

6. REFERENCES

[1] Detken, K.-O.: *Integrated Network Platform (INP) for Next Generation Networks (NGN)*; EURESCOM Summit 2002; Powerful Networks for Profitable Services; VDE Verlag GmbH; ISBN 3-8007-2727-7; Berlin 2002

[2] Detken, K.-O., Fikouras, I., Phillipopoulos, P.: *Service Discovery Integrated Network Platform*; Internetworking 2002; Australia 2002

[3] Guttman, E.: *Service Location Protocol: Automatic Discovery of IP Network Services*, IEEE Internet Computing, Vol. 3 Nr. 4, July 1999

[4] Mytanz, P.: *Service-Discovery unter der Vereinigung von UDDI und LDAP*; Diploma work at the University Bremen, Bremen 2004

[5] NOMAD Deliverable 1.2: *Report on Market and Business Model Survey*; European project NOMAD (Integrated Networks for Seamless and Transparent Service Discovery); July 2002; Brussels 2002

[6] NOMAD Deliverable 2.1: *NOMAD Specification*; European project NOMAD (Integrated Networks for Seamless and Transparent Service Discovery); November 2002; Brussels 2002

[7] NOMAD Deliverable 3.1: *Report on NOMAD Prototype System Development*; European project NOMAD (Integrated Networks for Seamless and Transparent Service Discovery); September 2002; Brussels 2003

[8] NOMAD Deliverable 3.5: *Service Discovery Middleware*; European project NOMAD (Integrated Networks for Seamless and Transparent Service Discovery); February 2004; Brussels 2004

[9] Renner, C., Bettstetter, C.: *A comparison of services discovery protocols and implementation of the service location protocol*, Institute of Communication Networks, Munich 2000

[10] Wahl, M., Howes, T., Kille, S.: *Lightweight Directory Access Protocol (v3)*, RFC-2251, December 1997