Service Discovery and Orchestration for Distributed Service Repositories

Ioannis Fikouras and Eugen Freiter

Bremen Institute of Industrial Technology and Applied Work Science (BIBA),
Hochschulring 20, 28359 Bremen, Germany
{fks,fre}@biba.uni-bremen.de
http://www.biba.uni-bremen.de

Abstract. Driven by the need for transparent discovery and orchestration of composite services out of elementary services, this paper introduces an innovative approach to Distributed Composite Services Orchestration and Discovery. The proposed concept is based on Variant Configuration theory and Distributed Service Repositories. The authors proceed to describe parts of the specification of a middleware platform based on the proposed concept capable of addressing the need for seamless discovery and composition of distributed services. This study was conducted as part of project NOMAD (IST-2001-33292) and presents the concepts behind the NOMAD Composite Service Configurator and its integration in the Distributed Service Repository.

1 Introduction

Significant technological advances in recent years in the areas of mobile devices and wireless communications were accompanied by infiltration of all aspects of our lives by all sorts of new Internet based services. Mobile communications and the Internet have been the two major drivers of consumer demand for new services in the last decade of the twentieth century [1].

Mobile phones are already pervasive in all major developed economies as well as in an increasing number of developing ones. The average mobile penetration in Europe in 2002 reached 72.4 per cent [2] and internet penetration in EU homes reached 38 per cent in December 2001 [3]. In November 2001, almost 50 per cent of the population over 15 years used the Internet either stationary or mobile. The rate of Internet take up by businesses is far higher at almost 90 per cent of enterprises with more than ten employees. Furthermore, it is forecasted that by 2005 an increasing portion of Internet users will be using wireless devices such as web-enabled cell phones and PDAs to go online and the number of worldwide Internet users will nearly triple to 1.17 billion [4]. By that time, a variety of different wireless network platforms with different properties, capable of transporting Internet traffic will be available [5]. In addition, the turn of operators towards license-free frequencies [6] and their eventual congestion will lead to the realisation of alternative dynamic network structures, namely Internet compatible, multi-hop, ad-hoc networks.

M.E. Orlowska et al. (Eds.): ICSOC 2003, LNCS 2910, pp. 59–74, 2003.

1.1 Problem Statement and Motivation

In spite of repeated past forecasts of the contrary, mobile Internet access today accounts for less than 10 per cent of those online globally even though the number of mobile users greatly exceeds the number of Internet-users [8]. However, it is expected that as early as 2005 over one half of the Internet population will consist of mobile access devices [4]. In countries with low Internet penetration wireless Internet devices will be the primary or only Internet access means.

The advent of the mobile Internet depends on the corresponding evolution of a new type of services vital for providing concrete added-value to users [9] resulting in motivation to adopt the new technology. In an environment with the potential for true global user mobility, a paradigm shift from stationary to mobility aware services and from a provider or operator-centric service model to a user-centric view [10] will be witnessed. Main characteristics of the new paradigm are:

- **User centricity**: Use of all available means to free the user from established restricting structures in order to offer the best possible service.
- **Mobility awareness**: Use of all data concerning the user's position, movement and direct environment or context for the provision of services.

A Composite Service Discovery technology addressing these issues was researched and developed by Project NOMAD (IST-2001-33292). This paper will present the concepts behind the NOMAD Composite Service Configurator and its integration in a Distributed Service Repository.

Sections 2 and 3 of this paper give an overview of existing approaches to Service Orchestration and Distributed Service Repositories. Section 4 proceeds to explain the NOMAD Composite Services data model while Section 5 describes how a configuration engine for Composite Services can be integrated in an LDAP repository. Section 6 provides the overall conclusions.

2 Approaches to Service Orchestration

Orchestration of Services is usually based on the concepts of Knowledge-based Variant Configuration. Case-based, rules-based and the Object Oriented approach including the concept of "Lean Configuration" are briefly illustrated in the following sections.

2.1 Knowledge-Based Variant Configuration

Knowledge-based Variant Configuration [18] is a process were complex products are composed out of elementary components. A Configurator is an expert system that supports this process and thereby uses predefined goals as well as expert knowledge. Design goals can be constraints, functional requirements, predetermined components or various quality criteria [19]. Such systems do not follow a single predefined method, but rather a strategy based on a series of small steps, each step representing a certain aspect or assumption leading to the configuration of the composite service.

Configuration is therefore considered as the solution to a single exercise and not the solution to a whole problem or problem class that has first to be methodically analysed. This implies the following:

- The set of all possible solutions is finite.
- The solution sought is not innovative, but rather is a subset of the available parts.
- The configuration problem is known and well defined.

2.2 Rules-Based Configuration

Existing configuration systems like JSR 94 [33] are based on knowledge bases consisting of a description of all the available components (objects) and an accompanying set of rules that define how specific objects should behave under defined conditions and thereby control the flow of configuration [24].

Consequently configuration rules are structured according to the "if-then" principle familiar from various programming languages. The "if" part of a rule contains the conditions under which the actions defined in the "then" part rule should be applied [30]. A configuration problem described using rules based concepts is composed of the following three elements [20]:

- Facts describe conditions that are necessary for configurations.
- Rules describe the relationships between facts and describe actions to be takes.
- Enquiries describe the problem to be solved.

The aim is to acquire answers to the questions posed with the help of the rules defined based on the known facts [20].

Rule-based systems are easy to implement due to the simplicity of the individual rules, but are hard to maintain after reaching a certain complexity. Production rules based systems are maintained by highly qualified knowledge engineers that must have considerable knowledge on the products in question and on the configurations system and most importantly the defined rules. Furthermore rule-based systems are usually restricted to a single knowledge domain in order to prevent the exponential complexity necessary for multiple rules-sets for multiple domains.

2.3 Case-Based Configuration

Case-based orchestration makes use of libraries containing similar problems and predefined solutions in order to formulate new composite services [34] thereby reducing the configuration problem to the following steps:

- The search for a similar case.
- The transformation of the original case to fit the current requirements.

The second step is where case-based configuration differs from other case-based methods, i.e. case-based reasoning or diagnosis where no such transformation is needed [32]. Collected knowledge can be used for further configuration under the following conditions:

- Creation and maintenance of appropriately organised libraries containing problems, solutions as well as the used process.

- Existence of algorithms and heuristics for the selection of appropriate cases from the library.
- The integration of case-knowledge in the configuration process. This includes procedures for checking case consistency and case transformation [31].

Configurations created on the basis of such a library can often be less efficient than others created with more conventional means. This is mainly due two the following characteristics of case-based methods:

- Resulting configurations are not fully compliant to the current requirements, but rather adapted products that were originally designed for different requirements.
- Changes in the knowledge domain can not be integrated in the case library without changing all relevant cases resulting in configurations that are sometimes not up-to-date [31].

2.4 Object-Oriented Variant Configuration

Object-oriented Variant Configuration is based on the concept of iterative composition of the final product out of a set of elementary components that have been previously organised according to a product data model into a structure, known as the object hierarchy that contains all knowledge related to the product in question. The relationships between components and how they fit together are described with the help of constraints.

Constraints are constructs connecting two unknown or variable components and their respective attributes, which have predefined values (taken from a specific knowledge domain). The constraint defines the values the variables are allowed to have, but also connects variables, and more importantly, defines the relationship between the two values [21]. In other words, constraints contain general rules that can be applied to make sure that specific components are put together in a correct fashion without having to specify any component-related rules or calculations [21]. The *constraint* satisfaction problem is defined as follows [22]:

- There is a finite set of variables $X = \{x_1, \ldots, x_n\}$.
- For each variable x_i, there exists a finite set D_i of possible values (its domain).
- There is also a set of constraints, which restrict the possible values that these variables are allowed to take at the same time.

The object hierarchy contains all relevant objects and the relationships between them in an "is-a" relationship that defines types of objects, object classes and subclasses, and their properties. The configuration process creates objects on the basis of this information according to the products being configured. In one specific hierarchy (as depicted in the following figure for the configuration of automobiles, classes for specific car types (i.e. coupé, minivan, etc.) are connected by "is-a" relationships to the main "car" class. This hierarchy also allows the breakdown of a product into components with the help of further "has-parts" relationships. These "has-parts" relationships are the basis for the decision-making process employed to create new configurations. An example of such a relationship would be the relationship between a chassis and a wheel. A chassis can be connected to up to four wheels in a passenger car, but the wheels are represented only once, with an appropriate cardinality (see Fig.1).

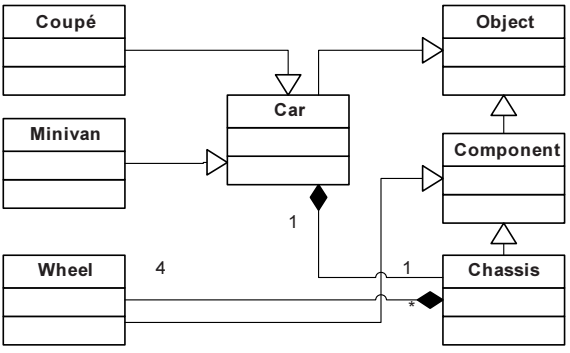

Fig. 1. Example object hierarchy of a specific product domain

The greatest hurdle to be resolved when creating new configurations is the fact that the software is required to make decisions that are not based on available information. Such an action can possibly lead to a dysfunctional composition or simply to a combination that does not conform to user requirements. In this case all related configuration steps have to be undone (backtracking) in order to return to a valid state. The longer it takes for the configuration to detect that a mistake has been made, the more difficult it is to correct the error in question [19]. The configuration process itself is composed of three phases [24]:

- Analysis of the product in order to define possible actions.
- Specification of further configuration actions.
- Execution of specified actions.

These actions are:

- *Disassembly of the product into its components.* This is meant to reduce the complexity of the problem and create a large number of smaller objectives in the manner of conventional top-down specification.
- *Assembly of components, integration and aggregation*. This step creates a product out of its components in a bottom-up manner.
- *Creation of specialised objects*. Object classes are specialised through the definition of subclasses.
- *Parameterise objects*. Define attributes and parameters for the specified objects that can be used for the application of constraints or other configuration mechanisms.

Object-oriented configuration is a modern approach to variant configuration suitable for complex structures in arbitrary product domains. Furthermore this approach allows for simplified maintenance of established service repositories through clear hierarchical structures.

2.5 Lean Configuration

The “Lean Configuration” [25] approach (developed in the course of the INTELLECT IST-1999-10375 Project) to variant configuration is object oriented, but reduces the configuration process to a search problem by eliminating the complex,

computationally intensive and error-prone first two steps of object oriented configuration thereby eliminating the need for back-tracking.

The reduction in complexity is realised by the usage of correctly configured, complete compositions as the basis for interactive configuration. As long as the user uses a pre-configured composition as a template for the new variant the configuration process can be transformed into a search problem and, specifically, a search for the next component to be exchanged. The Configurator supplies the user with lists of components that (a) are comparable to the service being exchanged and can be safely used in place of the component to be removed or (b) are compatible to existing services and can be safely added to the configuration. This mechanism ensures that the configuration is constantly in a correct state.

3 Distributed Service Repository Approaches

Discovery of a requested service can typically be accomplished in two separate manners:

- by directly contacting a known address that can supply the client with information on the available services as is currently implemented by UDDI[12].
- per broadcast. Broadcasts can be either focused on the local network or use mechanisms like multicast to reach a much larger group of service agents without the need for predefined addresses; an example for such protocols is Service Location Protocol (SLP) [13].

A major disadvantage of broadcast solutions is that they can produce enormous amounts of unnecessary traffic that grows exponentially with the number of hops (Time To Live, TTL) the broadcast is allowed to traverse (i.e. the number of networks it is allowed to flood). Furthermore such broadcasts are necessary every time the mobile node changes its environment (i.e. after a handover) or in some cases even every time a certain service is desired. Small TTLs on the other hand reduce the amount of signalling traffic generated, but coupled with the discrepancy between the networked and the physical world [14]; can lead to ineffective service discovery queries. Even clients and service providers in close physical proximity are not guaranteed to find each other due to possibly large “virtual” distances separating them in the Internet [14].

Centralised Service Repositories suffer from scalability issues that are usually addressed with the help of replication. Replicated repositories are however neither truly scalable, nor transparent due to the fact that updates occur only periodically. Existing technologies like UDDI v1 are considered to scale only moderately. The UDDI Replication and Scalability team is as of the writing of this document considering a distributed architecture for future revisions of its specification.

A distributed service repository based on a distributed directory like LDAP (Lightweight Director Access Protocol) can provide both a more scalable and more efficient solution. In addition to the performance benefits offered by LDAP, work within the NOMAD project is also progressing on an LDAP based distributed UDDI repository for providing compatibility with WebService technologies [15]. Within Project NOMAD an existing Free software UDDI repository (SOAPUDDI) is extended to support LDAP as a universal distributed backend to service repositories.

3.1 Lightweight Directory Access Protocol

In LDAP, directory entries are arranged in a hierarchical tree-like structure called the Directory Information Tree (DIT). Traditionally [16], 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 anything else. In addition, 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 the Relative Distinguished Name or RDN) and concatenating the names of its ancestor entries. For example, the entry for Ioannis Fikouras in the Internet naming example above has an RDN of uid=fks and a DN of uid=fks,ou=PPC,o=BIBA,c=DE".

Data on services stored in an LDAP distributed database can be spread across multiple LDAP servers. These servers are typically responsible for specific regions of the LDAP directory tree.

4 NOMAD Composite Services Data Model

This section introduces the NOMAD Composite Services Data Model starting with the NOMAD Mobility Aware Services Taxonomy and continuing with the various component and workflow types.

4.1 Mobility Aware Services Taxonomy

The following simple taxonomy is used as the basis for defining different types of Elementary Services and the relationships between them in the context of mobility aware Composite Services Configuration. This categorization is achieved mainly based on the type of functionality the Elementary Services offer. Services can thus belong to the following groups (see Fig.2):

- Stationary or Mobile
- Of limited availability or unlimited availability
- Information services

The defining attribute of stationary services is their fixed position. Such services are usable only by users in their immediate vicinity, as opposed to mobile Services that can change their location. An example for a stationary service would be a service provided in a physical store i.e. a haircut, a meal, etc., whereas mobile services could be taxis, couriers, etc that can change their location to meet the user. Mobile services can be of a logistical nature, but are not restricted to transportation services. In order to combine multiple stationary services into a composite service, logistics (mobile) services are required if both stationary services are in different physical locations. Combinations of multiple mobile services are also possible.

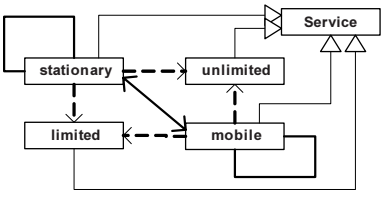

Fig. 2. NOMAD elementary services

Information Services are considered to be a special type of stationary services as they lack a number of characteristics thereby posing a smaller amount of requirements to the configuration process. An example for such a service could be an Internet based weather report system. This type of facility does not have a physical location, is nevertheless stationary, but does not require a mobile service to get connected to other services. Ubiquitous connectivity is assumed to be provided by networks. Networking facilities provided by the NOMAD integrated network platform [27] are assumed to be always available.

Services with limited availability are services that are provided based on the availability of finite resources. Such services usually require additional actions in order to handle reservations, prevent overbooking, etc. An example for such a service would be any facility that can accommodate a limited amount of customers (i.e. hotel, restaurant, etc.) Services that service customers on a FIFO basis and simply serve all incoming requests are considered to be services with an unlimited capacity. Both services with limited and unlimited capacities can be either stationary or mobile.

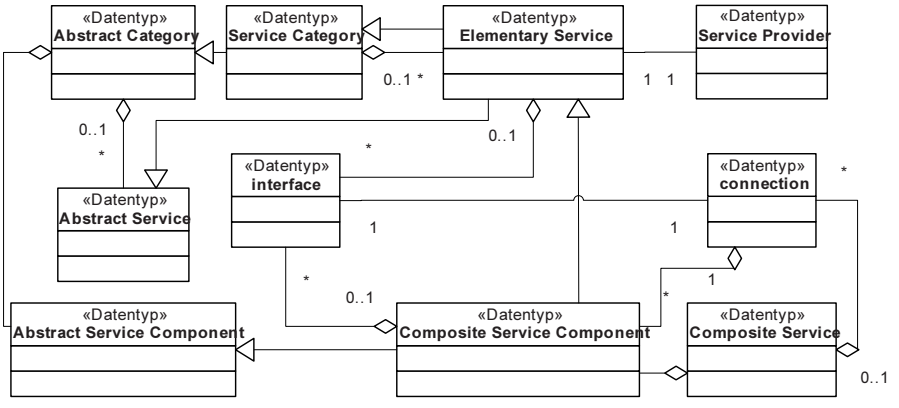

Fig. 3. Composite Services Data Model

4.2 NOMAD Composite Services Component Types

The NOMAD Composite Services Data Model divides services conceptually into two categories, *Elementary Services* and *Composite Services*. Elementary Services represent a specific instantiation of a service and contain all data needed to describe it. They inherit a set of general attributes available to all services from the *Abstract Service* data-type. *Composite Services* consist of groups of *Composite Service Com-*

ponents derived individually from Elementary Services. Composite Service Components inherit their attributes from Elementary Services, as well as attributes related to workflow management from the *Abstract Service Component* datatype. The purpose of these components is to describe the exact composition of the service, including data on which components are connected, by what *Interfaces* and in what order. Connections between such components are described additionally with the help of *Connection* components (see Fig.6).

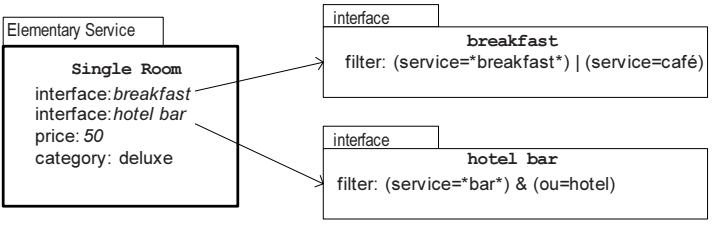

Fig. 4. Elementary Services with two Interfaces

Service Categories descend from the *Abstract Category* datatype and implement a means of grouping elementary services into sets according to functional criteria. These sets are meant to simplify and optimise the composite service configuration process, by providing predefined groups of components that can be used to reduce the amount of services the Configurator has to process in his/her search for suitable components.

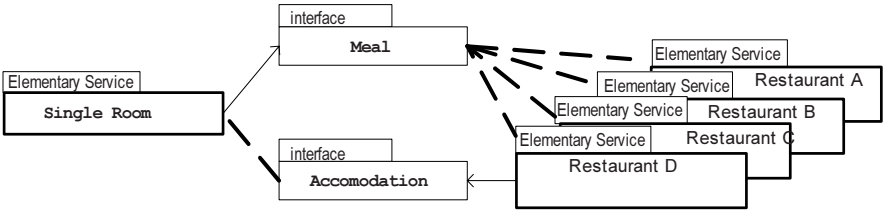

Fig. 5. Elementary Service "socket" (left) with multiple "plugs"(right)´

4.2.1 Composite Services, Interfaces, Connections, and Workflows

Interfaces between components implement constraints and as such offer mechanisms for determining whether Elementary Services are suitable for integration into a composite service (see Fig.6). The requirements that need to be fulfilled for a successful composition are derived both from Components connected to the Interface, as well as from user preferences or *Connection* components. Interfaces can be defined between Elementary Services, Composite Services, Service Categories and Service Providers and are a part of Elementary Services. Interfaces are mainly used to determine whether two Elementary Services fit, whereas Connection components describe a specific bond between two Composite Service Components.

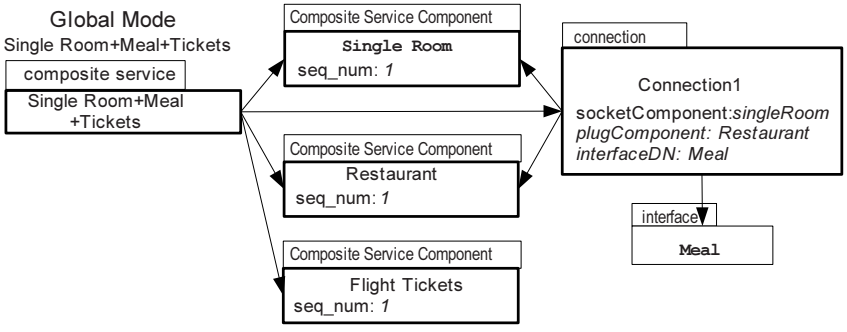

Fig. 6. Composite Service with Global Workflow Model

The relationship between interfaces and elementary services matched by the filters contained in an interface resembles the one between plugs and sockets, whereby interfaces as sockets match multiple plugs. Henceforth, connections to Elementary Components that have a direct reference to an interface via its unique identifier will be referred as "sockets" and components that are matched by a socket will be referred to as "plugs". An interface object is not restricted in its scope to use by only one pair of Service Components, but rather implements a generic rule (constraint) that can be used by multiple components for describing their interfaces.

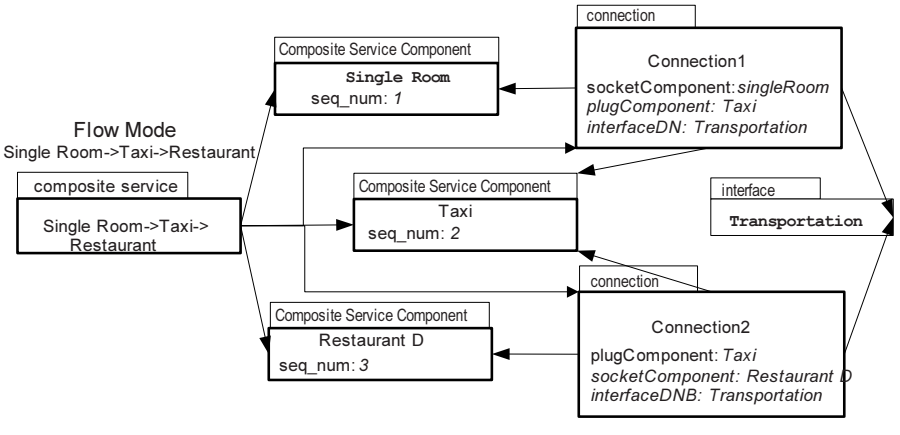

Fig. 7. Composite Service with Flow Model Workflow

An elementary component can be connected to multiple interfaces and can act as a plug and or a socket (see Fig.5) depending on whether it is executing a search for a compatible component (through one of its interfaces) or being the target of such a search. Provided that multiple interfaces are available, all plugs corresponding to the filters in the interface objects are possible composition candidates. A positive match between two components however requires matching interfaces in both directions. In

turn, this causes the Configurator to backtrack the connection between the plug and the socket in order to make sure that the plug also fits the socket.

When modifying existing Composite Services (i.e. when exchanging one Service Component for another) the Configurator needs to know precisely how components are connected to each other, to avoid altering the structure of the composition by using a different interface for connecting the new component with the rest of the composition. Connection components are therefore used for storing information on the connections between composite service and the corresponding interfaces (see Fig.7).

NOMAD Composite Services are assembled as a set of Composite Service Components arranged according to a specific type of workflow. Workflow functionality is introduced based on mechanisms specified by WSFL [26] (WebServices Flow Language) in order to ensure WebService compatibility. Composite Services can thus have one of the following types of workflow: *Flow Model, Global Model* and *Recursive Composition*.

- A Flow Model is a linear workflow where each service has to be executed in a specific sequence. The correct sequence of execution is stored within the Composite Service Components.
- Global Models provide a description of how the composed services interact with each other. This type of workflow requires no additional considerations.
- Recursive composition of services provides scalability to the composition language and support for top-down progressive refinement design as well as for bottom-up aggregation. Recursive composition of services is made possible by the loosely couple nature of NOMAD Composite Services. New Composite Services can be composed out of existing compositions by merging the existing groups of components into new bigger compositions.

5 Composite Services Schema and Engine Integration

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 [17] 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.

Middleware for the management and configuration of composite services can be thus integrated as an additional backend. This backend would be 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 be implemented locally as supporting module for the local Service Discovery node or be installed centrally.

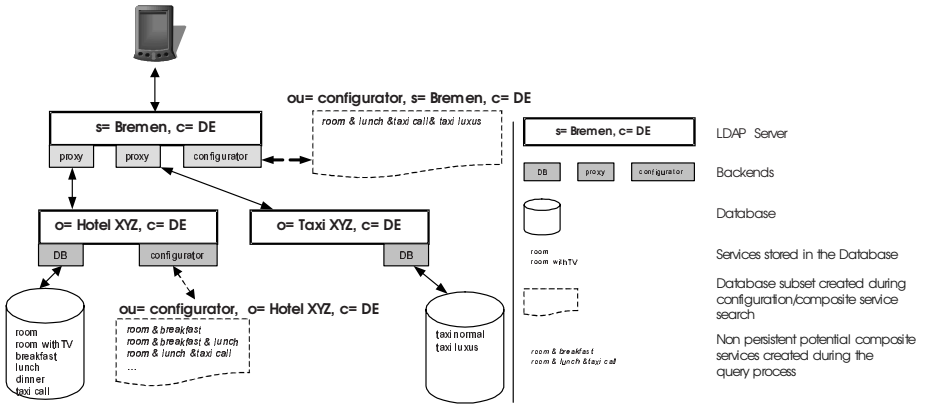

Fig. 8. Distributed Directory hierarchy with multiple Configurator Engines

A 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 for standards conform access 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 for recursive creation of composite services, were a Configurator can consult any number of other Engines providing some subset of the overall composite service.

The Configurator handling Composite Services for a Service Repository in the area of Bremen (see Fig.8) could for instance be set-up as one of the back-ends for a Directory Server with the distinguished name (DN) "s=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, s=Bremen, c=DE) and are forwarded to the Configurator back-end.

5.1 LDAP Composite Services Schema

The domain specific knowledge required for Service Composition is hard-coded into an LDAP schema [28]. Elementary Service types that are to be part of the composition process are described in this schema. LDAP schemata are Object Oriented and consist of definitions of Object Classes defined as a collection of attributes with clearly defined datatypes. Object Classes may inherit attributes from multiple other such classes. The Composite Services schema thus defines the attributes describing the services, as well as the interfaces required for service composition. Component attributes are statically defined in the LDAP schema of the directory server and have either a MUST (compulsory) or MAY (optional) status. Individual instantiations of

the services may only vary in their choice of attributes, and possibly additional optional attributes not used for composition purposes.

The following code is a simplified representation of an LDAP v3 schema with respective object definition in LDIF [29] format for Composite Service Components, Composite Services, Interface and Connection components.

```
objectclass (1.1.4.2.1 NAME 'serviceComponent'

DESC 'Composite Service Component'

SUP elementaryService, abstractServiceComponent

MUST (seqNumber $ elementaryServiceDN )

MAY interfaceDN )

dn: cn=flight,cn=Fly&Drive, …

objectclass: abstractServiceComponent

objectclass: elementaryService

objectclass: serviceComponent

elementaryServiceDN: cn=flight,ou=KLM,o=Transport, …

seqNumber: 1

dn: cn=RentAcar, cn=Fly&Drive, …

objectclass: abstractServiceComponent

objectclass: elementaryService

objectclass: serviceComponent

elementaryServiceDN: cn=rentAcar,ou=AVIS,o=Transport, …

interfaceDN: cn=AVIS_interface,cn=Fly&Drive, …

seqNumber: 2
```

Service Components inherit attributes from the Abstract Service Component and the Elementary Service, such attributes include the sequence number used for implementing the workflow. Furthermore a Service Component contains a direct connection to the Elementary Service it represents and its related interfaces.

```
objectclass (1.1.1.2.1 NAME 'Interface'

DESC 'Interface object'

MUST (cn $ filter)

MAY availability )

dn: cn=AVIS_interface,cn=Fly&Drive, …

objectclass: interface

filter: &((objectclass=flight)(destination=Bremen)
(class=first))
```

The actual task of defining the relationship between two elementary services is accomplished by Interface objects. Interfaces are separate objects that contain LDAP filters or DNs describing the compatible components, as well as additional attributes specifically related to the interface and are linked to a specific component via their Distinguished Name (DN), a unique identifier positioning the object within the LDAP Directory Information Tree (DIT).

LDAP filters defined in an Interface object effectively represent preconditions that have to be met in order to achieve a working composition. Valid operators used for constraint resolution are all the operators supported by the LDAP filter specification [23] and include basic operations like “equal”, “not equal”, “greater than” and “less than”. The filters contained in an interface object make use of standardised attributes defined in the schema for the required type of service. The use of non-standardised attributes is possible, but may lead to ineffective queries.

```
objectclass (1.1.5.2.1 NAME 'connection'

DESC 'Connection in Composite Service'

MUST (cn $ plugComponent $ socketComponent $ interface)
)

dn: cn=Fly&Drive_Connection,cn=Fly&Drive, …

objectclass: connection

plugComponent: cn=FlyTicket,cn=Fly&Drive, …

socketComponent: cn=CarRental,cn=Fly&Drive, …

interface:cn=AVIS_interface,cn=Fly&Drive, …
```

Connection objects describe a specific connection between two Service Components. As such these objects contain the unique identifier of the plug and the socket components involved as well as the interface describing compatible Elementary Services.

6 Conclusions

This paper has shown the need for transparent discovery and orchestration of composite services out of elementary services. Furthermore an approach was illustrated for Distributed Composite Services Orchestration and Discovery based on Variant Configuration theory and a Distributed Service Repository. The authors propose the implementation of a middleware platform capable of addressing the issues identified and proceed to describe parts of its specification.

Acknowledgements. Project NOMAD (IST-2001-33292) is funded by the European Commission within the IST Programme of the FP5. The authors wish to express their gratitude and appreciation to the European Commission and all NOMAD partners for their strong support and valuable contribution during the various activities presented in this paper.

References

1. ITU Internet Reports 2002: Internet for a Mobile Generation, International Telecommunication Union, September 2002, http://www.itu.int/osg/spu/publications/sales/mobileinternet/
2. Mobile and internet penetration rates increase 08/08/2002, http://www.europemedia.net/
3. eEurope Benchmarking Report, European Commission, http://europa.eu.int/information_society/eeurope/benchmarking/index_en.htm
4. eTForecasts, Internet Users Will Surpass 1 Billion in 2005, http://www.etforecasts.com/pr/pr201.htm
5. Niebert, N, "Convergence of Cellular and Broadband Networks towards Future Wireless Generations", In Wireless Strategic Initiative (WSI) Book of Visions 2000 – Visions of the Wireless World Workshop, Brussels 2000
6. Mohr, W., "Alternative Vorschläge zur Spektrumsnutzung für IMT-2000/UMTS", Spektrumsworkshop ITU-R, October 2000, Geneva, Switzerland
7. Katz, HR, Brewer, AE, "The Case for Wireless Overlay 'Networks", In: SPIE Multimedia and Networking Conference (MMNC'96), January 1996, San Jose, CA, USA
8. Keryer, P. (2000), Presentation at the workshop: Visions of the Wireless World, 12th December 2000, Brussels
9. Pöyry, P., Repokari, L., Fournogerakis, P., Fikouras, I., "User Requirements for Seamless and Transparent Service Discovery", In: Proceedings of eChallenges 2003, 22–24 October 2003, Bologna, Italy, to be published
10. Fikouras, I., Wunram, M., Weber, F., "Seamless Integration of Mobile Products and Services – User-centricity and Mobility Awareness for mCommerce", In: Proceedings of the Wireless World Research Forum (WWRF) Kick-off meeting, Munich 2001
11. Gilder, G., "Telecosm : How Infinite Bandwidth Will Revolutionize Our World", Free Press, September 11, 2000
12. Universal Description, Discovery and Integration of WebServices (UDDI), http://www.uddi.org/
13. Guttman, E., Perkins C., Veizades J. and Day M., "Service Location Protocol, Version 2", RFC 2608, June 1999

14. Ioannis Fikouras, "Peer-to-peer Service Engineering for Integrated Networks", In: Wireless Technology 2002 Business Briefing, World Markets Research Centre, London, UK, Pages 88–91
15. B. Bergeson, K. Boogert, “LDAP Schema for UDDI” draft-bergeson-uddi-ldap-schema-01.txt, May 2002
16. RFC2253 "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names.
17. http://www.OpenLDAP.net
18. Tank, W., Wissensbasiertes Konfigurieren: Ein Überblick. Künstliche Intelligenz (KI) 1 1993
19. Neumann B. (1988) Configuration expert systems: a case study and tutorial. Proc. Conf. on AI in manufacturing, Assembly, and Robotics, Oldenbourg
20. Lunze, J. Künstliche Intelligenz für Ingenieure. München, Wien: Oldenburg Verl. 1994
21. Tsang, E.P.K., Foundations of Constraint Satisfaction, Academic Press, London and San Diego, 1993
22. Barták, R., in Proceedings of Week of Doctoral Students (WDS99), Part IV, MatFyzPress, Prague, June 1999, pp. 555–564
23. Howes, T., “The String Representation of LDAP Search Filters” RFC 2254, December 1997
24. Cunis R., Günter A., Strecker H. (1991) Begriffshierarchie-orientierte Kontrolle. In Das PLACON-Buch. Informatik Fachberichte Nr. 266. Springer, Berlin, Heidelberg
25. Fikouras, I., Detken, K., Lean Configuration: Interactive 3D Configuration for E-Commerce Environments, In: J. Gasos, K-D. Thoben (Eds.), "E-Business Applications: Technologies for Tomorrow's Solutions", Springer, Berlin, 2002
26. Prof. Dr. Frank Leymann, WebServices Flow Language (WSFL 1.0), May 2001, www.ibm.com/software/solutions/ webservices
27. Koojana Kuladinithi, Andreas Könsgen, Stefan Aust, Nikolaus Fikouras, Carmelita Görg Ioannis Fikouras, "Mobility Management for an Integrated Network Platform", 4th IEEE Conference on Mobile and Wireless Communications Networks, Stockholm September 2002
28. M. Wahl, T. Howes, S. Kille, “Lightweight Directory Access Protocol (v3)” RFC2251-2256, 2829–2831, December 1997
29. G. Good, “The LDAP Data Interchange Format (LDIF) - Technical Specification” RFC2849, June 2000
30. Bense, H; Bodrow, W. Wissensbasierte Dialogführung für ein Beratungssystem zum Softwarequalitätsmanagement. In Objektorientierte und regelbasierte Wissensverarbeitung. Heidelberg, Berlin, Oxford: Spektrum, Akad. Verl., 1995
31. Cunis, R; Günter, A; Strecker, H. Fallbasiertes Konstruieren mit Bibliothekslösungen. In Das PLACON-Buch. Springer, Informatik Fachberichte Nr. 266 1991
32. Günter, A; Dörner, H; Gläser, H; Neumann, B; Posthoff, C; Sebastian, H-J. Das Projekt PROCON: Problemspezifische Werkzeuge für die wissensbasierte Konfigurierung. Technische Uni Chemnitz, Martin-Luther Uni Halle-Wittenberg, Uni Hamburg, Technische Hochschule Leipzig, Technische Hochschule Zwickau. PROCON-Bericht Nr.1, 1991
33. Alex Toussaint, BEA Systems, Java Specification Requests, Java Rule Engine API , http://www.jcp.org/en/jsr/detail?id=94
34. Limthanmaphon, B. and Zhang, Y. (2003). Web Service Composition with Case-Based Reasoning. In Proc. Fourteenth Australasian Database Conference (ADC2003), Adelaide, Australia. Conferences in Research and Practice in Information Technology, 17. Schewe, K.-D. and Zhou, X., Eds., ACS. 201–208.