Author(s): Jian, Yu; Falcarin, Paolo; Rego, S.; Ordas, I.; Martins, E. ; Quan, Sun; Trapero, R.; Sheng, Q.Z.
Title: XDM-Compatible Service Repository for User-Centric Service Creation and Discovery
Year of publication: 2009
Citation: Jian, Y. *et al*. (2009) 'XDM-Compatible Service Repository for User-Centric Service Creation and Discovery' *IEEE International Conference on Web Services, (ICWS 2009),* Los Angeles, CA, 6-10 July 2009, IEEE, pp. 992-999
Link to published version: http://dx.doi.org/10.1109/ICWS.2009.155

2009 IEEE International Conference on Web Services

XDM-Compatible Service Repository for User-Centric Service Creation and Discovery

Jian Yu[1], Paolo Falcarin[2], Sancho Rego[3], Isabel Ordas[4], Eduardo Martins[5], Quan Sun[2], Ruben Trapero[6] and Quan Z. Sheng[1]
[1]School of Computer Science, The University of Adelaide, Adelaide, Australia
{jian.yu01,qsheng}@adelaide.edu.au
[2]Dip di Automatica e Informatica, Politecnico di Torino, Torino, Italy
{paolo.falcarin, quan.sun}@polito.it
[3]Portugal Telecom Inovação, Aveiro, Portual
sancho-c-rego@ptinovacao.pt
[4]Téléfonica I+D, Madrid, Spain
ioa@tid.es
[5]RedHat-Jboss
emartins@redhat.com
[6]DIT, Universidad Politécnica de Madrid, Madrid, Spain
rubentb@dit.upm.es

***Abstract*—The key objective of OPUCE system is to enable the participation of end-users in the management of their own services, by providing them with innovative tools which allow an easy creation and delivery of personalized communication and information services. This paper describes the OPUCE service and component repository, which extends the OMA OSPE service model storage approach XDM. By integrating an ebXML Registry using the native notification mechanisms of XDM, the search capability of the repository is dramatically improved. Moreover, this repository also exploits semantic Web technology to provide an intuitive visualized browser for convenient service exploring.**

Keywords-service repository; XDM; user-centricity; semantic services; visual discovery

I. INTRODUCTION

In recent years the needs of the users concerning telecommunications services, especially mobile ones, have evolved rapidly, requiring the collaboration of an increasing number of network resources of various technologies and user data. On the other hand, users have been excluded from current service provisioning models, also not having any way to personalize their services according to their needs.

But now, a new horizon awaits users. It is time for the Telco world to follow the Web 2.0 paradigm, where participants actively create and share contents among themselves, going for its own Telco 2.0, in which users become *prosumers* (producers + consumers) of services.

In this context, the challenge is a converged Web of information technologies and communications services where users could design their own highly personalized integrated services, relieving operators from the pressure of service development and publishing. The European Union sponsored IST-FP6 integrated research project OPUCE [1] (Open Platform for User-Centric Service Creation and Execution) was conceived to achieve this goal.

Focusing on the promising notion of *prosumer*, OPUCE aims at developing a full-fledged platform that enables end-users, even technically non-experienced ones, to create, manage, deliver, and share their personalized services covering both information technology and telecommunications features, regardless of the infrastructure or technology that lies beneath.

The service repository is an indispensible part of a service platform such as OPUCE. It should accomplish several requirements so that the platform can work smoothly: an effective way to manage the different parts which compose the service description, a notification system to get informed when changes happen and an structured classification of services to ease the search functionalities. General-purposed service registries such as UDDI [1] and ebXML Registry [2] are not suitable to fulfill those requirements. In OPUCE, we propose another approach which tries to solve the drawbacks of existing solutions:

- the repository supports a faceted approach to access OPUCE service descriptions, which means we can manage (retrieve, delete etc.) one aspect of the service description without affecting the other parts in a fine-grained manner (Section 2 describes the OPUCE service description language in detail);
- Open Mobile Alliance (OMA)2 specification XML Document Management (XDM) [3] based server is used as the remote interface for the service repository

[1] http://www.opuce.eu/
[2] http://www.openmobilealliance.org/

978-0-7695-3709-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICWS.2009.155

to enable data management on mobile devices and automatic XML validation;

- an ebXML Registry server is used as the service query and discovery interface to provide advanced search capabilities
- the semantic Web technology is leveraged to provide a visualized service browser to facilitate end users exploring services.

The rest of the paper is organized as follows: Section 2 introduces the OPUCE service description language. Section 3 describes the general architecture of OPUCE service repository, including its interfaces, the integration between XDM server and ebXML Registry interfaces and how to map the faceted service descriptions to the repository. Section 4 presents the semantically enhanced visual service browser. Section 5 discusses related work and finally we conclude the paper in Section 6.

II. OPUCE SERVICE DESCRIPTION LANGUAGE

A proper execution and management of user-created services requires a way to describe all the aspects that characterize a service. It is not only required the typical description of the most common aspects of a service (*What is the name of the service? Who is the creator? What is this service for?*), but also specific ones (*Which components does this service use? How does it use them? What tasks are required at deployment time?*). Furthermore, the platform also needs to know all the aspects that completely describe the available components. It is useful (and mandatory) to be able to answer the questions *"How can this component be combined with other ones?" "What operations can be done with this component?"* or *"How to configure this component to do what I expect it to do?"*

In OPUCE we have created a complete service description language [4] that describes all the required aspects of a service, to be used both for composed services (in the following simply called "services") and for base services (called "components").

One component provides some functionality which can be considered atomic from the point of view of the user. One service is composed of at least one component

When a service specification is mapped to a specific composed service, it is generated a set of service description documents which represent the service in all its aspects.This service description is represented in a set of XML documents linked together. These files are called facets, each addressing a different aspect of the service.

A facet is an abstraction over one or more service properties that provide a partial description of a service. Each facet addresses a focused task of a service consumer, describing functional, non-functional, or management properties.

The main advantages of the faceted approach are the modularity and the flexibility [5]. New facets can be added as long as it is necessary. Furthermore, each facet can be described by using a different language, either standard or customized.

Figure 1 depicts a graphical representation of the OPUCE service specification.

The figure on the left represents the master description of the service, which includes general purpose information of the service, such as the service name (in different natural languages), the version, the creator's name, or the unique identifier for the service. This master document also contains a list of the facets included in the service description. Each facet contains its type and a link to the document where the complete facet description is.

The right side of the figure represents the structure of a facet description document. Facet description documents are saved in separate files, so they need an identifier that must be the same as that of the service they belong to. The facet specification language contains the format of the language used in the facet specification data. For instance, the facet representing the logic of the composition is described by using the BPEL language [6], thus the facet specification language contains a reference to the BPEL language and the facet specification data contains the BPEL script, which can be executed in the OPUCE service execution environment [7].

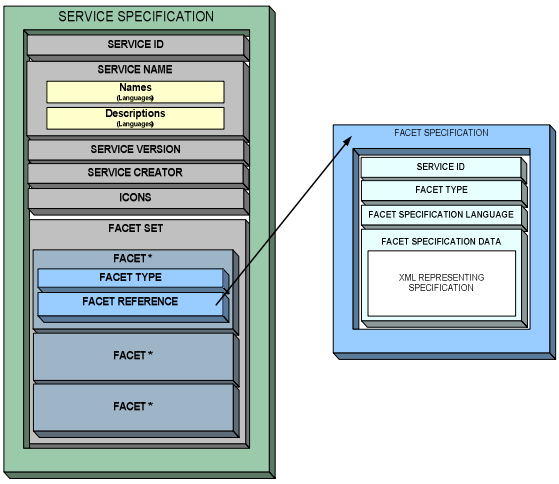

Figure 1. Service Specification with Facets

Apart from the Logic Facet, additional facets have been used in OPUCE, such as a Deployment Facet, a Semantic Facet, an Interface Facet or an Accounting Facet. The usage of all the facets defined in OPUCE is not mandatory for both types of services (services and components). For instance, the logic facet is mandatory for a composed service but not for a component (since we don't care which the internals of a component are). For sake of completion these other facets have been defined: the Provisioning Facet, SLA Facet, Graphical Facet, Logic Facet, Behavior Facet, Context Awareness Facet, Requirements Facet, Deployment Facet, Scheduling Facet and Lifecycle Facet.

III. SERVICE REPOSITORY ARCHITECTURE

The OPUCE service and component repository is a storage/repository for all the descriptor facets of the OPUCE services and components. It should provide both basic keyword-based and advanced semantics-aware search functionality to other modules of the OPUCE platform like Service Creation Environment (used by the user to create and compose the own services) and Service Advertising (used to notify user about updates on services of interest).

This crucial requirement drives towards the subdivision of the service repository module into two parts: the Service Storage to manage the push/pull functionalities of the services and Service Registry to define the metadata and built the service taxonomy.

Figure 2 shows an external view of the architecture of the OPUCE Service Repository, with more details on the interfaces and the OPUCE modules that use them. As we can see, a Registry module and a Repository module (the aforementioned Service Storage) compose the overall repository, which provides four groups of interfaces:

- Store interfaces: provide basic database-like functionalities including Create /Retrieve/ Update/ Delete of services and components.
- Search interfaces: provide basic keyword-based and advanced semantics-aware search for stored services and components.
- Notification: provide notification to interested parties when some parts of the service/component specification are changed.
- Management: provide metadata, especially classification and version management functions.

The Store interfaces are divided into Service Store and Component Store. The Service Store provides basic database functionality to the services; the user can create/retrieve/update/delete a service and its various facets. With the exception of the notification mechanism which is provided by a SIP event framework, all the interfaces are collectively presented as Web Services to other modules.

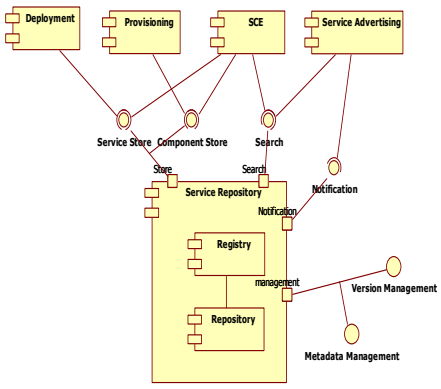

Figure 2. Service Repository: External View

Component Store provides basic database functionality to components; much like the Service Store does to services. Likewise, it also provides interfaces to change the status of a component with respect to the service lifecycle (created, deployed, active, inactive, etc)

The Search interface provides keyword-based and tag-based service and component search functions, and it also provides various search functions which are required by Service Lifecycle Management, and Service Creation Environment and other modules: examples are search by event, by owner, by deployment status or by underlying technology of a component.

The Notification interface provides basic change subscription/notification functionality. Users can subscribe to a specific change of components and services.

The Metadata Management interface deals with the metadata, specially the classification metadata, of services and components. Users can build/modify a classification scheme, which is a tree of classification nodes. Every node on the scheme tree represents a concept, which may be associated to a service or component. These associations are created based on the semantic facet of a service or component. An apparent application of the scheme would be the service and component portal interface: a classified directory can be obtained directly from a classification scheme.

The Service Storage acts as a common database, taking care of the storage, retrieval, updating and deletion of services. The Service Registry, on the other hand, manages the metadata of the services in storage; this metadata could be keywords, tags, classifications, relations, and formal ontologies.

Concerning the implementation platform for the repository, common database management systems were found not to satisfy the requirements for their weak metadata management capability.

The ebXML Registry standard fulfils the main requirement of the OPUCE platform: it is a registry as well as a repository. The ebXML Registry has a powerful Registry Information Model (RIM), which provides an upper ontology for defining the service metadata. The most important information models in RIM include:

- *Classification Information Model* to define classes that enable classification;
- *Association Information Model* to define the associations between concepts;
- *Service Information Model* to define classes that enable service description.

The strength of this RIM is the reason why this technology has been chosen for the Registry part of the OPUCE Service & Component Repository, allowing a versatile, generic and easily expandable semantic taxonomy to be built. The repository functions of ebXML Registry are delegated to a database management system, which works as a plug-in of the registry. However, the

Repository part of the ebXML Registry was found not to be the best solution for the Service & Component Repository because of its slow performance as pertains the requirement parameters of OPUCE.

The Service Storage was achieved through the use of an XML Document Management (XDM) server, as defined in OMA specification [3]. This server is responsible for handling the management of user XML documents stored on the network side, such as presence authorization rules, contact and group lists (also known as resource lists) and static presence information. Since the storing of a component in the repository is, in essence, storing the XML documents of the facets, XDM is an ideal match, in particular because it also offers strong XML schema validation of all stored XML documents.

Furthermore, the inherent subscription/notification mechanisms of this technology provide the natural implementation of this capability in the Service Repository.

A. Service storage: XDM Server

Figure 3 depicts the XDM Server arquitecture. In a short summary, the XDM Server is an XCAP Server that also provides a SIP [8] Event framework interface to subscribe changes in documents managed by the server.

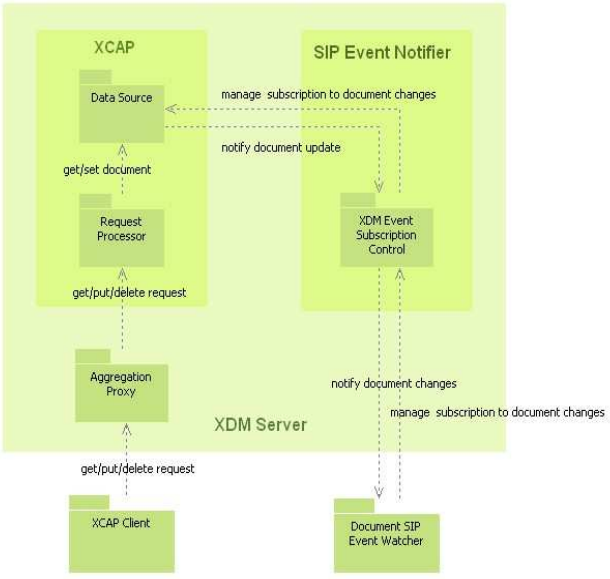

Figure 3. XDM Server Architecture

In OPUCE architecture the XDM Server is used to store and manage the Service Repository XML facets. Each of those facets is associated with an application. In order for an application to use those resources, application specific conventions must be specified. Those conventions include the XML schema that defines the structure and constraints of the data, well-known URIs to bootstrap access to the data, and so on. Each of those application-specific conventions for an XML document type, in this case OPUCE XML facet, is an XCAP Application Usage. Specifically, an XCAP Application usage defines:

- Application Unique ID (AUID) - the ID used in XCAP URIs to point to a specific XCAP Application Usage.
- Default Document Namespace – an XCAP URI may have a section, using an XPath expression, which selects a specific element/attribute in a XML document stored in the server. In those XPath expressions the namespace of elements/attributes is defined using prefixes, the default document namespace defines the XML Namespace of elements/attributes without prefix in those URIs.
- MIME Type – the MIME Type used when exchanging XML content XML Schema and Data Constraints - the XML Schema to validate documents; Data constraints, which are impossible to validate with XML Schema, e.g. one element value must be a ISO country name that belongs to Europe
- Data Semantics - semantic definition on documents content, used by applications filling data, not validated by servers
- Naming Conventions - what is the document name for each user? Are there global documents under a specific name?
- Resource Interdependencies - one request may update other documents as well, e.g. global/index document in rls-services, a composition of all users/*/index <service/> elements
- Authorization Policies - what each user can read or write? If not specified the default policy is considered, on which a user is allowed to read and modify its own documents, and read global documents.

The XDM Server has a notification mechanism that follows OMA standard XDM. The simplified interface offers operations to

- *subscribeDocument*: add a new subscription to changes in the specified XML document, returning a Response code that indicate failure or success (including subscription id);
- *subscribeAppUsage* : add a new subscription to changes in all XML documents for the XCAP Application Usage with the specified AUID;

Related unsubscribe operations are obviously available.

B. Mapping Faceted Descriptions into Repository

When a service or component facet is introduced in the repository, there are two internal steps undertaken. The first is the actual storing of the facet; the second is parsing it for the needed metadata. As a facet is an XML document, it is stored in the XDM server via the available XCAP interface.

The URI of the document, which is the first parameter in the above interface function, is composed by three parameters:

- Application Unique ID - it refers essentially to the type of facet being introduced. The concept was explained before, but suffice it to say that each facet allowed in the repository must have an associated Application Unique ID so that proper XML schema validation may occur;
- DocumentParent – it can be considered as a "folder" inside a certain application usage. It will either be "component" or "service".
- DocumentName – the name of the component or service for which we are storing a facet.

These three parameters uniquely identify a facet inside the repository. Upon submission to the XDM server, the facet will be validated against its application usage. This ensures that only facets that fulfil a previously, well defined XML-schema will be accepted into the repository; in this manner, the service repository becomes protected from invalid and erroneous descriptors which could later compromise its capabilities. Once a facet is validated and correctly inserted, it will be passed to a parser which deals with introducing the metadata in the ebXML Registry.

Building the service's taxonomy follows a similar process in all facets. Each Service or Component is represented inside the ebXML Registry by an object which is an "instance" of a certain "Concept", namely that of "Service".

The parser will search for the relevant elements in the introduced document, according to facet type. Each element, for example a keyword in the semantic facet, corresponds to an object instance of the "Keyword" concept. The particular instance related to this element will then be obtained from the repository if it already exists or created if it doesn't. Afterwards, all that is left is to create a specific type of Association between this concept object and the service object that represents the component/service.

Using this process, the OPUCE Registry is populated with an internal taxonomy which is essentially composed of objects, each an instance of a certain abstract concept such as Keyword or Service or Person, connected to each other in a network of associations. This powerful and flexible metadata structure easily allows one to perform various searches and associations.

IV. SEMANTICALLY ENHANCED SERVICE BROWSER

In such a service platform like OPUCE in which users need to seek for the components that are available to build their services, a semantically enhanced service browser is extremely useful. It should provide to the user the taxonomy of services as it has been built in the Service Registry. The present section explains how the metadata structure has been provided in OPUCE.

Using ontology to represent semantic information of services, a rich and considerable accurate meta-information about services can be used to locate desired services from a large bunch with high precision. Using some advanced semantics visualization techniques [9], we can enable the end-user to search services in a graphically browsing manner without using any complex query language (e.g. SPARQL [10]) for writing a query statement. The implementation of this feature highly pronounces the user-centric theme of OPUCE.

In the case of applying semantic web technologies to discovering/searching services, it is necessary to distinguish descriptions about individual services from general knowledge of the domain that services reside in. Using the terms of Description Logics (DLs), the theoretical foundation of semantic Web technology, descriptions on individual services are the ABox (assertion box), and general domain knowledge belongs to the TBox (Terminology box).

While TBox contains sentences describing concept hierarchies and relations, ABox contains "ground" sentences stating where in the hierarchy individuals belong (i.e., relations between individuals and concepts). The OPUCE Ontology stores the TBox of the general domain knowledge and service semantic facet is the ABox about individual services.

Following reasons make us separate TBox and individual service assertions in the context of OPUCE:

- Improved performance on reasoning. To check whether a service belongs to a class, we only need to reason on TBox and the assertions of this service, without reasoning on other individual service assertions.
- Modeling, managing, and changing to TBox and service assertions can be made independently.

Based on above statements, in OPUCE, we created a single knowledge base, i.e. TBox. The assertions about a specific service, with references to the knowledge base, are kept in the semantic facet of this service.

As for the technologies employed, OWL-DL [11] is the language to describe the knowledge base. For semantic reasoners, Pellet [12], an open source DL Reasoner in Java implementing the common DIG Interface [13] is used, in order to allow interoperability with other reasoners sharing this interface specification based on a concept language and a minimal set of operations.

Figure 4 describes the architecture of the service browser. The OWL Manager module, which is built on top of OWL API [14], loads the general domain knowledge from OPUCE Ontology and a hierarchical structure in ClusterMap format to display and query. After a new OWL ontology is added, only one location in the hierarchical structure is instantiated. The DL Reasoner reasons through the "OPUCE Ontology" and finds all the possible locations the service can reside in the structure

(mostly the ancestor node) and updates the corresponding nodes which represent the added service to the structure.

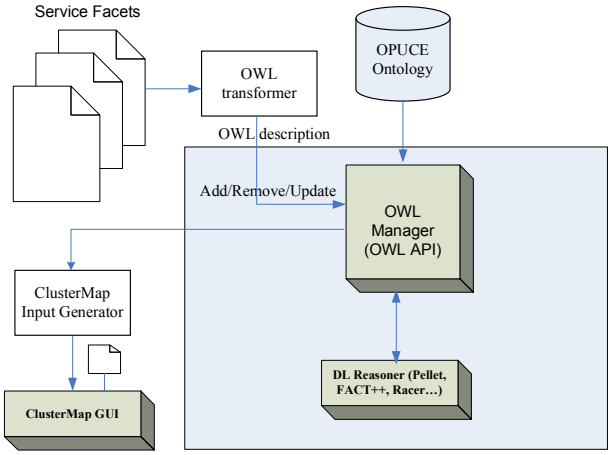

Figure 4. Architecture of the visual semantic ervice browser

The browser uses the OWL Manager to manipulate OWL elements, which has an interface for integration with the reasoners. The DL reasoner provides standard and cutting-edge reasoning services for OWL ontologies.

The ClusterMap GUI contains information visualization technology developed at Aduna [3] for visualizing sets of classified objects. Its main purpose is to show if and how these sets overlap (see Figure 5).

It can create visualizations of collections of hierarchically classified objects and uses XML file as input. Therefore Cluster Map is very suitable and well designed to display the hierarchical structure as mentioned above.

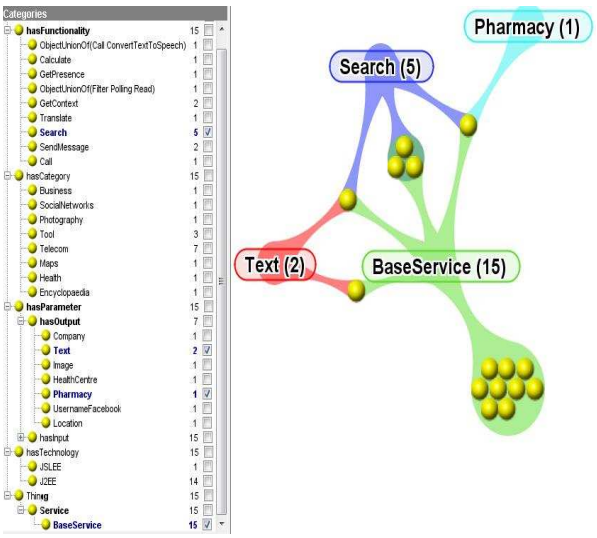

Figure 5. Browsing Services with Aduna ClusterMap Viewer

[3] http://www.aduna-software.com/

The left area of Figure 5 is the hierarchical structure of the general domain knowledge (only those which have nodes are displayed). The number following the name is the number of the services with the property.

On the right area, the service with combined properties can be easily identified by the balls that are surrounded with different colors. For example, there are three services which are both "BaseService" denoted by cyan color and "hasFunctionality Search" denoted by blue color and otherwise do not belong to a semantic category. From the user's point of view, instead of writing query commands, only some clicks in the check boxes are needed to retrieve the information.

A. The OPUCE Services Ontology

The semantic facet is the part of the service description most directly accessed by the end-users as it is a descriptive representation of the service mainly designed to ease service search and discovery specially aimed at service composition.

However this part of semantic facet does not always exist in service descriptions, thus a transformer which extracts, analyzes and generates the semantic facets is presented in the architecture. We use OWL language as semantic description instead of plain XML, and thus an OPUCE service ontology (see Figure 6 for detail) has been created to define some OWL classes used to manage the meta-data. This meta-data allows the categorization of OPUCE services in order to semantically model the hierarchy of all the services.

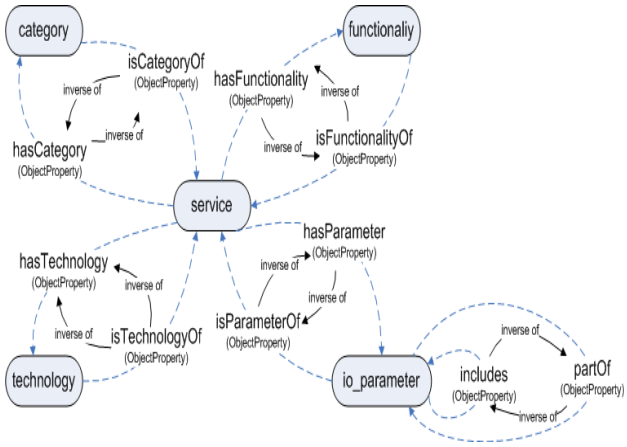

Figure 6. OWL Classes in OPUCE ontology

These OWL classes are:

- *service* class, which is the parent class of all the OPUCE services;
- *category* class, which defines the domain of the offered service (e.g. telecom, sports, tourism, etc);
- *functionality* class, which gives information about what the service does (e.g. send_message, call, polling, etc);

- *technology* class, which specifies the technology being used to implement the service;
- *IO_Parameter* class, which contains the set of parameters that can be input and/or output of a service;

The service class has two sub-classes, *base_service* class and *composite_service* class, which represent components and services respectively.

The OPUCE services are subclasses of either *base_service* or *composite_service* classes and have several object properties associated, which are instances of *category*, *functionality*, *technology* and *IO_parameter* classes, to define the semantic description of the service.

B. Semantic Facet Mapping

Since a facet is a projection over one or more service aspects that provide a specific description of a service, hopefully we can retrieve semantic information from the provided service facets files automatically. The OPUCE Service description must be converted in OWL to include the following information of a service: category, technology, functionality, and I/O parameters. As this ontology file does not exist when a service is created, we need a transformer to extract, analyze and generate it from service facets. In fact, not all ontology information can be extracted from current service facets so we need users to input the information during service creation.

The most important and difficult part is how to map the information given in several different service facets files to the field in the OPUCE ontology:

- *io_parameter*: I/O information can be found in Interface facets of services. Actions describe the set of operations the component offers, which is composed of synchronous and asynchronous actions. Actions may have input properties that have to be set before the invocation and may also fire events.
  In the following example of SendSMS component:
  <In>to</In>
  <In>content</In>
  <Out>SMSSentEvent</Out>
  <Out>SMSNotSentEvent</Out>
  Two inputs and two outputs are shown. Their contents are defined in the previous part of this interface facet file. If the input or output *io_parameter* cannot be found in the OPUCE Ontology, we have to delete it in our generated OWL files. For example, we don't care about the SMSSentEvent and SMSNotSentEvent and they are not in the TBox, so no output part of SendSMS is presented in the OWL file. Meanwhile we may have to modify the OPUCE ontology files to give a better match result in order not to miss some semantic information.
- *technology*: the Component Provisioning facets contain the information for service components (Base Services). We can find the technology information from the "componentType" element.
- *Category*: in the plain XML file of semantic facet description, there is a "Keywords" field which is assigned by the creator to help to classify the service into a predefined category tree. In the case SendSMS it is "messaging".
- *Functionality*: it's not easy to get the functionalities of a service from service facets for the notion "functionality" is abstract and difficult to be automatically interpreted by machines. We can get this information from the service descriptions in master facet, or set it manually, especially for Base Services.

The Cluster Map Viewer operates on XML-structured data files, describing the visualized objects, the class tree and the sets of objects contained by the classes.

An XML data file is rooted by a Classification Tree element containing the following two information types:

1. The ObjectSet: a part describing all information objects displayed in the visualization. Every Object has an ID attribute that will be needed later on for the definition of the Classifications. In our case each newly added service is modeled as an object in this object set with its name and ID.

2. The ClassificationSet: a part describing the class hierarchy or taxonomy. The classes in the taxonomy are defined inside the ClassificationSet element, using a list of Classification elements.

Every Classification has an ID that is used to register parent-child relationships between other classes. In our case "Calculate" has a parent which is "hasFunctionality" and named as *c0*. In the displayed graph, the "Calculate" will be shown as the child class of "hasFunctionality" class. The aim of the OWL reasoner in our case is to find all the possible parent classes of a specified class given the whole hierarchy of the ontology knowledge denoted in the OPUCE Ontology files. After finding all the parent classes, the OWL API adds the ID of the service in the ObjectIDs of the corresponding classification.

V. DISCUSSION AND CONCLUSION

With the rising and prevalent of service-oriented computing, providing suitable service repositories for the convenient retrieving, discovery, and management of service descriptions also gains a momentum. UDDI [1] and ebXML Registry [2] are the two main specifications for managing and discovering e-service descriptions. To support flexible service description management, we extend ebXML Registry with new facet-based functionalities. As we discussed in the first Section, the main application area of UDDI and ebXML Registry is Web information systems and lack the support to telecommunication protocols such as SIP. To enable description management on mobile devices, a XDM server [3] is used to integrate with ebXML Registry.

Extending service repositories with semantic features to enhance the discovery capability has been a hot topic.

In [15], the authors enrich ebXML Registry with OWL ontologies. In [16-18], various approaches are used to enhance UDDI with semantics. In our approach, service semantic descriptions are used to visualize the classification information of services, resulting in an intuitive service browser facilitating service exploration.

In this paper, we describe the technical details of a novel service repository used in the user-centric service platform OPUCE. Technologies from the mobile computing, Web information systems and semantic Web fields are integrated to fulfill the complex requirements aroused in the platform. Currently, the OPUCE platform has been integrated into the mobile open source community Open movilforum[4], and a usability testing plan is underway.

VI. ACKNOWLEDGMENT

The work presented in this paper was performed as part of the OPUCE project and partly funded by the European Union under contract IST-034101, as an Integrated Project of the 6th Framework Programme, Priority IST. The authors want to thank all project partners for their valuable contributions, and in particular Alvaro Martinez Reol and Albeto Leon (Téléfonica).

REFERENCES

[1] L. Clement, A. Hately, C. von Riegen, T. Rogers (eds.), UDDI Version 3.0.2 Specification, UDDI Specification Committee Oct. 2004.

[2] K. Breininger, F. Najmi, and N. Stojanovic (eds.), The ebXML Registry Repository Version 3.0.1, OASIS, Feb. 2007.

[3] OMA (Open Mobile Alliance) standard XDM, RFC 3265(Session Initiation Protocol (SIP)-Specific Event Notification) and IETF draft ,An Extensible Markup Language (XML) Configuration Access Protocol (XCAP) Diff Event Package, June 2006.

[4] J.C. Yelmo, R. Trapero, J. M. del Alamo, J. Sienel, M. Drewniok, I. Ordás, and K. McCallum, "User-Driven Service Lifecycle Management – Adopting Internet Paradigms in Telecom Services", 5th International Conference on Service-Oriented Computing (ICSOC'07), Springer, Sept. 2007, pp. 342-352.

[5] J. Walkerdine, J. Hutchinson, P. Sawyer, G. Dobson, and V. Onditi, "A Faceted approach to Service Specification", 2nd IEEE International Conference on Internet and Web Applications and Services (ICIW'07), May 2007, pp. 20-20.

[6] A. Alves, et al (Ed.), Web Services Business Process Execution Language Version 2.0, OASIS WSBPEL TC, Jan 2007, available online at http://docs.oasis-open.org/ wsbpel/2.0/.

[7] C. Baladrón, J. Aguiar, B. Carro, J. Sienel, R. Trapero, J.C. Yelmo, J.M. Del Álamo, J. Yu., P. Falcarin, "Service Discovery Suite for User-Centric Service Creation", Service Oriented Computing: a look at the Inside (SOC@Inside'07) workshop, Vienna, Sept 2007, pp. 46-51.

[8] SIP, Session Initiation Protocol. On-line at http://www.ietf.org/rfc/rfc3261.txt

[9] Aduna Cluster Map Viewer Autofocus, On-line at http://www.aduna-software.com/technologies/autofocus

[10] SPARQL (SPARQL Protocol and RDF Query Language). On-line at http://www.w3.org/TR/rdf-sparql-query/

[11] M.K. Simith, C. Welty, and D.L. McGuiness (eds.), OWL Web Ontology Language Guide, W3C Recommendation. Online at http://www.w3.org/TR/owl-guide/

[12] Pellet, the open source OWL Reasoner. On-line at http://clarkparsia.com/pellet

[13] The DIG Interface (Description logic Implementation Group). On-line at http://dl.kr.org/dig/

[14] OWL API. On-line at http:// owlapi.sourceforge.net/

[15] A. Dogac, Y. Kabak, Gokce B. Laleci. "Enriching ebXML Registries with OWL Ontologies for Efficient Service Discovery", 14th International Workshop on Research Issues on Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE'04) , pp. 69-76.

[16] M. Paolucci, T. Kawamura, T.R. Payne, K. Sycara, "Semantic Matching of Web Service Capabilities", Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, Springer, Heidelberg, 2002, pp. 485-511.

[17] R. Akkiraju, R. Goodwin, P. Doshi, S. Roeder, "A method for semantically enhancing the service discovery capabilities of UDDI", Workshop on Information Integration on the Web (IIWeb 2003), Acapulco, Mexico, August 2003.

[18] D. Kourtesis and I. Paraskakis, "Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery", S. Bechhofer et al.(eds.): ESWC2008, LNCS 5021, 2008, pp.614-628.

[4] http://opuceportal.movilforum.com/