Data, Information and Process integration with Semantic Web Services

DIP

Data, Information and Process Integration with Semantic Web Services

FP6 – 507483

Deliverable

WP4: Service Usage

D4.19

QoS-enabled Service Discovery Component Prototype 2 Report

Le-Hung Vu (EPFL), Fabio Porto (EPFL), Othman Tajmouati (EPFL), Sebastian Gerlach (EPFL)

December 18, 2006

Summary

In the upcoming Semantic Web and service oriented architectures, Quality of Service (QoS) is among the most important searching and ranking criteria influencing the user in the selection of a service among several functionally equivalent ones. This report includes the installation instruction and a brief description of the *second (and final) prototype* of the QoS-enabled Semantic Web service discovery component, following the specification in the Deliverable D4.17 [4].

The QoS-enabled Semantic Web service discovery is the process of automatically finding Web services that fulfill a certain user goal in terms of their quality of service (QoS) criteria. Typically, users express a goal by specifying their functional and QoS requirements the Web services should provide in order to achieve it. These requirement specification can either be written by the users given his prior knowledge of the WSML language or be generated automatically from dedicated graphical user interfaces of the search engine. We extend goals and Web service descriptions to support the specification of QoS parameters and provide a discovery component which is capable of combining both QoS and functionality-based service discovery into one integrated module.

Regarding the development of exploitable tools, this report and its associated QoS discovery prototype has the following contributions:

- It implements the most important functionalities of a QoS discovery component: (1) select the services fulfilling user's QoS requirements by doing the semantic matchmaking between a list of services against the submitted user goal; (2) perform the service ranking given various user's preferences and estimated QoS values of the services from the reports of reputable users; (3) parallelize the most expensive steps of the discovery process to enable high scalability.

- The developed component could be used in two ways: (1) as a stand-alone service discovery application which includes the capability of the functionality-based service discovery component; (2) as a discovery module to be plugged-in to another system, for example, the WSMX or the WSMO Studio framework. The implementation of our QoS discovery prototype conforms to the WSMX/DIP API, making it be easy to use for the interested DIP partners and be interoperable with the other DIP tools such as the WSMO Studio.

- A parallel query processing system, named CoDIMS, is used to parallelize the operations of the QoS Discovery component. The CoDIMS-D (for Discovery) system leverages the discovery functionality to cope with thousands of Web service descriptions and multiple user requests.

- We also provide a list of WSMO ontologies: upper ontologies for QoS discovery and ranking algorithms, full-fledged working WSML Web service and goal descriptions, as well as the dedicated QoS and ranking ontologies for each developed example. These are useful for the demonstration of the modeling of QoS requirements and offerings in various realistic application scenarios.

Therefore, this report and its associated QoS discovery prototype are relevant for the following audience: the use case partners, the WSMO, WSML and WSMX developing groups, the developers and IT experts who are interested in technological

solutions for Semantic Web service discovery based on QoS and/or non-functional properties criteria. In the DIP framework, this report can be of the interests of the following partners:

- WP1 - to consider the extensions of the WSMO model to support QoS parameter modeling more explicitly.
- WP2 - requirements for the repository interface for retrieving Web service descriptions and ontologies as input parameters for the discovery process.
- WP8 - use case partner defining a B2B Telecom case study with QoS.
- WP10 - use case partner defining an e-banking case study with QoS.
- Other partners interesting in QoS-based Semantic Web service discovery and its applications.

Disclaimer: The DIP Consortium is proprietary. There is no warranty for the accuracy or completeness of the information, text, graphics, links or other items contained within this material. This document represents the common view of the consortium and does not necessarily reflect the view of the individual partners.

Document Information

IST Project Number	FP6 – 507483	**Acronym**	DIP
Full Title	Data, Information, and Process Integration with Semantic Web Services		
Project URL	`http://dip.semanticweb.org/`		
Document URL			
EU Project Officer	Werner Janusch		

Deliverable	**Number**	4.19	**Title**	QoS-enabled Service Discovery Component Prototype 2 Report
Work Package	**Number**	4	**Title**	Service Usage

Date of Delivery	**Contractual**	31-Dec-2006	**Actual**	31-Dec-2006
Status	version 0.1		final □	
Nature	prototype □ report ☒ dissemination □ ontology □			
Dissemination Level	public ☒ consortium □			

Authors (Partner)	Le-Hung Vu (EPFL), Fabio Porto (EPFL), Othman Tajmouati (EPFL), Sebastian Gerlach (EPFL)			
Resp. Author	Le-Hung Vu, Fabio Porto		**E-mail**	lehung.vu@epfl.ch fabio.porto@epfl.ch
	Partner	EPFL	**Phone**	+41 (21) 693-7573,+41.21.693.52.53

Abstract (for dissemination)	This report includes the installation instruction and description of the implemented features of the final prototype of the QoS-enabled discovery component.
Keywords	Semantic Web service, SWS, service discovery, QoS, Goal,API

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
30-11-2006	1	Le-Hung Vu	First draft
1-12-2006	2	Le-Hung Vu, Fabio Porto, Othman Tajmouati	v 1.0 - First complete version of the report

Reviewers				
	Ozelin López		**E-mail**	ozelin@isoco.com
	Partner	ISOCO, Spain	**Phone**	
	Maciej Zaremba		**E-mail**	maciej.zaremba@deri.org
	Partner	DERI Galway, Ireland	**Phone**	

Project Consortium Information

Partner	**Acronym**	**Contact**
National University of Galway	NUIG	Dr. Sigurd Harand Digital Enterprise Research Institute (DERI) National University of Ireland, Galway Galway Ireland E-mail: sigurd.harand@deri.org Tel: +353 91 495112
Fundacion De La Innovacion.Bankinter	Bankinter	Monica Martinez Montes Fundacion de la Innovation. BankInter, Paseo Castellana, 29 28046 Madrid, Spain Email: mmtnez@bankinter.es Tel: 916234238
British Telecommunications Plc.	BT	Dr. John Davies BT Exact (Orion Floor 5 pp12) Adastral Park Martlesham Ipswich IP5 3RE, United Kingdom Email: john.nj.davies@bt.com Tel: +44 1473 609583
Swiss Federal Institute of Technology, Lausanne	EPFL	Prof. Karl Aberer Distributed Information Systems Laboratory École Polytechnique Féderale de Lausanne Bât. PSE-A 1015 Lausanne, Switzerland E-mail : Karl.Aberer@epfl.ch Tel: +41 21 693 4679
Essex County Council	Essex	Mary Rowlatt, Essex County Council, PO Box 11, County Hall, Duke Street, Chelmsford, Essex, CM1 1LX, United Kingdom. E-mail: maryr@essexcc.gov.uk Tel: +44 (0)1245 436524
Forschungszentrum Informatik	FZI	Andreas Abecker Forschungszentrum Informatik Haid-und-Neu Strasse 10-14 76131 Karlsruhe Germany E-mail: abecker@fzi.de Tel: +49 721 96540
Institut für Informatik, Leopold-Franzens Universität Innsbruck	UIBK	Prof. Dieter Fensel Institute of computer science University of Innsbruck Technikerstr. 25 A-6020 Innsbruck, Austria Email: dieter.fensel@deri.org Tel: +43 512 5076485

ILOG SA	ILOG	Christian de Sainte Marie 9 Rue de Verdun, 94253, Gentilly, France E-mail: csma@ilog.fr Tel: +33 1 49082981
inubit AG	inubit	Torsten Schmale, inubit AG, Lützowstraße 105-106 D-10785 Berlin, Germany E-mail: ts@inubit.com Tel: +49 30726112 0
Intelligent Software Components, S.A.	iSOCO	Dr. V. Richard Benjamins, Director R&D Intelligent Software Components, S.A. Pedro de Valdivia 10 28006 Madrid, Spain E-mail: rbenjamins@isoco.com Tel. +34 913 349 797
MDR Partners	MDR	Rob Davies, MDR Partners, 8 St. Andrew Street, Hertford, Herts., United Kingdom, SG14 1JA, Email: rob.davies@mdrpartners.com Tel. +44 (0)208 8763121
Hanival Internet Services GmbH	HANIVAL	Alexander Wahler, Hanival Internet Services GmbH, Kirchengasse 13/1a A-1070 Wien Email: wahler@niwa.at Tel. +43(0)1 3195843-11
The Open University	OU	Dr. John Domingue Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, MK7 6AA, UK E-mail: j.b.domingue@open.ac.uk Tel.: +44 1908 655014
SAP AG	SAP	Dr. Elmar Dorner SAP Research, CEC Karlsruhe SAP AG Vincenz-Priessnitz-Str. 1 76131 Karlsruhe, Germany E-mail: elmar.dorner@sap.com Tel: +49 721 6902 31
Sirma AI Ltd.	Sirma	Atanas Kiryakov, Ontotext Lab, - Sirma AI EAD, Office Express IT Centre, 3rd Floor 135 Tzarigradsko Chausse, Sofia 1784, Bulgaria E-mail: atanas.kiryakov@sirma.bg Tel.: +359 2 9768 303

Unicorn Solution Ltd.	Unicorn	Jeff Eisenberg Unicorn Solutions Ltd, Malcha Technology Park 1 Jerusalem 96951, Israel E-mail: Jeff.Eisenberg@unicorn.com Tel.: +972 2 6491111
Vrije Universiteit Brussel	VUB	Pieter De Leenheer, Starlab- VUB Vrije Universiteit Brussel Pleinlaan 2, G-10 1050 Brussel, Belgium E-mail: Pieter.De.Leenheer@vub.ac.be Tel.: +32 (0) 2 629 3749

List of Keywords/Abbreviations

- CoDIMS - Configurable Data Integration Middleware System framework.
- CoDIMS-D - Configurable Data Integration Middleware System for Discovery.
- DBMS - Database Management System.
- DHT - Distributed Hash Table.
- EAI- Enterprise Application Integration.
- NFP - Non-Functional properties.
- P2P - Peer-to-Peer.
- QoS - Quality of Service.
- NFPs - Non-Functional Properties.
- SWS - Semantic Web service.
- UDDI - Universal Description, Discovery and Integration protocol.
- WSD - Web service Description.
- WSMO - Web Service Model Ontology.
- wsmo4j - WSMO API for Java.
- WSML - Web Service Model Language.
- WSMX - Web Service Execution Environment.
- QML - Quality of service Modeling Language.
- WSLA - Web Service Level Agreement.
- WSOL - Web Service Offering Language.

Table of Contents

List of Figures

1 Installation and Usage of the QoS-enabled Discovery Component

1.1 Installation Guide for Windows-based Systems

1.1.1 Quick Installation Instructions

For impatient readers, the simplest and fastest way to run the QoS discovery component is:

- download the whole compressed bundle available at `http://lsirpeople.epfl.ch/lhvu/download/qosdisc/qosdisc2.zip`.
- unzip the above file into a local directory and modify the *qosdisc.properties* file as instructed by the inline comments therein.
- run the *createDB.bat* file to initialize and populate the database with the available service descriptions. This would take about 50 to 60 seconds depending on your computer configuration.
- open a shell console and run the shell script file *run.bat*.

1.1.2 Downloading the Necessary Files

The main download page for the QoS discovery component is at: `http://lsirpeople.epfl.ch/lhvu/download/qosdisc/`. From this starting point you can find the links to all other related documents.

The QoS discovery component is available at: `http://lsirpeople.epfl.ch/lhvu/download/qosdisc/qosdisc2.zip`. The following files and directories are contained in the downloaded archive:

- the file *qosdisc2.wsmx*: a JAR file of the binary executables of the QoS-enabled discovery component. This can be used separately or added into a WSMX installation as a component of the WSMX framework.
- the file *qosdisc.properties*: the configuration file of the QoS discovery component.
- the file *InputSettings.csv*: the configuration file of the reputation-based QoS estimation library.
- the file *COPYING*: the detailed copyright information.
- the file *createDB.bat*: a shell script file to create the underlying database and initialize it with the available service descriptions.
- the file *run.bat*: a script file for running the component in stand-alone mode.
- the directory *ontologies*: the WSMO test suite (WSMO goals, services, ontologies) for testing and using the QoS discovery component.
- the directory *lib*: all necessary libraries to use the QoS discovery component.

- the directory *dbinit*: the SQL scripts to create/clear the database appropriately.
- the directory *codims-home*: the configuration files of the CoDIMS-D query processing system.
- the directory *examplecode*: the example (Java) code listings for developers who want to interface with the QoS discovery component.

For convenient, in the following description we assume that the user downloads and installs the QoS discovery component in his/her local directory `C:/Temp/qosdisc2`.

In the subdirectory *lib* of the archive, the user can find the following necessary libraries:

- The library for the functionality discovery component *funcdisclite.jar*. Currently, this is the implementation of a light-weight semantic matchmaker, as specified in Deliverable D4.14.
- The QoS reputation management library: *lhvu-qos-rep.jar*.
- Ostermiller Java Utils package (version 1.4.03): *ostermillerutils_1_04_03_for_java_1_4.jar*.
- Apache mathematic package (version 1.0): *commons-math-1.0.jar*.
- Derby DBMS libraries (release 10.1.3.1): *derby.jar*, *derbyclient.jar*, *derbynet.jar*, and *derbytools.jar*.
- CODIMS-D query processing systems library: *codimsd.jar*.
- Apache Axis package *axis.jar* (release 1.4 1855 April 22 2006).
- JAX-RPC (version 1.1): *jaxrpc.jar*.
- KAON2 Reasoning engine: *kaon2-2005-11-14.jar*.
- Log4J library: *log4j-1.2.13.jar*.
- WSML reasoner wrapper: *wsml2reasoner-20060522.jar*.
- WSML parser library: *wsmlparser-20060210.jar*.
- WSMO4j 0.5.2: *wsmo4j-0.5.2.jar*.
- WSMO API 0.5.2 library: *wsmo-api-0.5.2.jar*.
- WSMX integration API (*wsmx-integration-API-2006.jar*).

1.1.3 Configuring the QoS-enabled Discovery Component

The *qosdisc.properties* file contains the following configurable parameters for the QoS discovery component:

- Property *installdir*: to be set to the directory where the user installs the component, for example, `C:/Temp/qosdisc2`.

- Property *goal*: URI of the WSMO goal containing the QoS requirements of the user, to be matched again the services in the DBMS. The goal descriptions should also be semantically annotated with the QoS requirements, as in the provided example goals in the default *qosdisc.properties* file.

- Property *output*: local path name of the directory to produce the output file containing the description of the ontological instances which describe the returned ranking values, for example, one can use the same installation directory `C:/Temp/qosdisc2`.

- Property *functional*: should be set to *true* or *false* depending on whether we want the QoS discovery component to call the functionality-based discovery component or not.

- Property *inputsettings*: the place of the configuration settings for reputation-based QoS estimation operators. This is usually the same directory as the *installdir* property.

- Property *codims-home*: is the path where the query processing system CoDIMS should look for itself. This is usually the subdirectory *codims-home* of the *installdir* diretory, for example, *codimshome*=`C:/Temp/qosdisc2/codims-home`.

- Property *ranking* contains the URI of the ontology to define the base concepts of the ranking algorithms. Similarly the property *comparison* is the URI of the ontology to define the comparison between QoS instances (should be the QoS upper ontology). Users are supposed not to modify these parameters.

- Property *wsmxhost*: URI of the WSMX host entry point for testing.

- Properties *db.driver*, *db.protocol*, and *db.name* are the configuration of the way of the discovery component should look for the Derby DBMS. Normally these settings can be ignored (commented out) since a Derby server will be started internally within the discovery component during its running time. In case the user wants to test the parallelized discovery or wants to use another separate Derby DBMS network server, these properties must be reconfigured appropriately.

- Property *startserver* specifies whether the user wants the QoS discovery component to automatically start the Derby DBMS server itself. This should be kept as the default value *true*, unless the user wants to use his or her own Derby network server, as aforementioned.

The last part of the properties file is the configuration for various loggers of the discovery component. During the testing, one may need to set some loggers to the DEBUG/WARN level in order to turn/off the details information about the discovery process, e.g., *log4j.logger.ch.epfl.qosdisc.operators.ReasoningContext=DEBUG*. The user can also reconfigure the loggers to print out the result/debug/info messages to a log file instead of the console.

IMPORTANT NOTES:

- The URIs of the goals, services, and ontologies can be a remote identifier like: *goal*=`http://lsirpeople.epfl.ch/lhvu/ontologies/Lite/Goal0.wsml` or a local path name like *goal*=`file:///C:/Temp/qosdisc2/ontologies/Lite/Goal0.wsml`.
- The pathname of the file and diretories in the configuration file should follow the convention C:/Temp/qosdisc2, i.e., it uses the symbol / to separate the directories instead of the \.
- A user should pay attention to save the above files to his or her computer with their *original* names. Some browsers, e.g., Microsoft Internet Explorer, have the tendency to automatically save a file under the new name with a default extension according to the file type, e.g., the file *qosdisc.wsmx* may be saved under the name *qosdisc.zip*, which makes thing more confusing.

1.2 Running the QoS-enabled Discovery Component

1.2.1 Running the Component in Stand-alone Mode

After the installation and deployment of the QoS-enabled discovery component as described in Sections 1.1.1 (or 1.3), a user can run and test the component in the stand-alone mode by:

- configuring the *qosdisc.properties* to suit his/her needs, e.g., specify the goal and the list of Web service descriptions you are going to work with.
- opening a DOS (or UNIX) console and run the file *run.bat*.

The result of the QoS discovery process will be displayed in the console (by default) or written into a log file according to the user's configuration of the logger in the *qosdisc.properties* file.

1.2.2 Running the Component in Graphical Mode

We have also implemented a dedicated Web-based GUI for the component, which is accessible from the Demonstration section of the download page `http://lsirpeople.epfl.ch/lhvu/download/qosdisc/`. The user can browse the service repository, load a new service file, enter input to generate a goal automatically and perform the discovery interactively. After the execution, one can browse the result set via this user interface.

The Web-based GUI is self-explanatory, so we do not provide the detailed usage guide of the component for this graphical mode.

1.2.3 Developer's Guide to Interface with the Component

For developers who want to interface with the QoS-enabled discovery component themselves, the following example code is provided:

- *LoadDatabaseIntegratedDemo.java*: shows how to create the DBMS and populate the Derby DBMS with appropriate data, e.g., service descriptions, user reports, etc.
- *TestStandaloneDatabase.java*: illustrates how to interface with the discovery component in a stand-alone fashion. This may be of interest for the users who want to integrate the QoS-enabled discovery component with their application, for example, WSMO Studio.

The above files are in the *examplecode* directory of the complete download bundle *qosdisc2.zip* and also available from the main download page of the component (section Documentation).

1.3 Installation Guide for UNIX-based Systems

The installation instruction for UNIX-like systems is mostly the same as in Section 1.1.1. The main difference is that one should change the permission of the files *createDB.bat* and *run.bat* to ”executable” appropriately before running them. This can be done with the UNIX command *chmod u+x createDB.bat run.bat.*

2 Installation and Usage of the CoDIMS Framework

In this chapter, we explain:

- How to use CoDIMS from the available QoS-Discovery Web interface.
- How to install your own QoS-Discovery server.
- How to use CoDIMS from Java code.
- How to prepare a Grid environment.

If you find a problem in the installation, please contact us. We are also pleased to receive feedbacks and comments.

2.1 CoDIMS Overview

CoDIMS (Configurable Data Integration Middleware System) is a middleware environment for the generation of adaptable and configurable data integration middleware systems. Data integration systems were designed to provide an integrated global view of data and programs published by heterogeneous and distributed data sources. Applications benefit from these types of system by transparently accessing resources independently of their localization, data model and original data structure.

We derived from CoDIMS a Configurable Query Processing Engine, i.e. a framework where users could define and execute their requests. Afterwards, we improved the framework to run QoS-Discovery requests. We call this new framework CoDIMS-Discovery or CoDIMS-D, which is the new version of CoDIMS adapted to the QoS-Discovery component (for brevity reasons we term it CoDIMS in this document). Refer to the user manual of CoDIMS [5] for a complete description of the framework.

2.2 Use CoDIMS from the Web interface

In order to run a parallel discovery query, a user would load the `http://codims.epfl.ch/qosdisc` page. The latter presents a list of available Web Services registered in the internal repository. This may take some time for loading depending on the number of registered Web Services. After that, you need to do the following:

- Click on *Discovery* tab.
- Go to *Parallel* Demo.
- Choose a goal definition to be run. By default, the goal is retrieved from the file `file:///C:/Progra~1/Apache~1/Tomcat~1.5/webapps/qosdisc/WEB-INF/classes/ontologies/Lite/Goal0.wsml`, which corresponds to the location of a WSML goal on the server. You can change this goal by entering the URL of your own WSML goal definition, for instance: `http://www.dip.com/QoSDiscovery/myGoalDefinition.wsml`. Note however that the goal will continue to run against the list of known Web service descriptions.

- Click *Run* to begin the execution.

An explanation of the execution scneario is in order. In this QoS-based parallel Discovery demo, users look for available File Hosting Services. A given goal (*Goal0.wsml*) defines users desires by specifying three QoS parameters: *UploadSpeed*, *Availability*, *MaxDownTime*, and one environmental condition *NetworkBandwidth* [4]. The query can be modeled as a conjunction such as:

$Goal(x) = UploadSpeed(x, 750, Kbps) \wedge Availability(x, 0.99, Percentage) \wedge MaxDownTime(x, 1, Minute) \wedge NetworkBandwidth(x, 100, Mbps)$.

The goal also specifies weighting information to be used by the ranking algorithm. The discovery function is applied over a repository containing 45 Web services. Once evaluated, selected Web service descriptions are ranked and presented, with their ranking value, at the bottom of the window. Their corresponding descriptions can be obtained by clicking over the links associated to individual Web service labels.

During the execution, you will see a bar chart representing the progression of the execution. The chart displays at the bottom of each bar a number representing a remote node id used for parallelization. When a remote node processes a tuple; a blue rectangle is drawn on the respective column. The numbers in the top dynamically inform users about the execution evolution by showing the total number of tuples (Web service descriptions) processed so far by each node. The execution elapsed-time is proportional to the number of initial Web Services and to the number and configuration of available remote nodes in the distributed environment.

At the end of the execution, the list of Web Services is displayed with the ranking score. Each Web Service is annotated with a green or red square, depending on whether it has accepted (it matches the goal) or rejected.

2.3 Use CoDIMS within Tomcat

2.3.1 System and software requirements

The following installation instructions are to be applied to the central node, i.e. the node to host a Tomcat server. The QoS-Discovery component has been tested under Windows XP, UNIX and LINUX operating systems; and unless we detail the difference, the installation is the same for the 3 operating systems.

You should first install on your local node Java 1.5 and Tomcat 5.5. The latter's installation directory is referred to *TOMCAT_HOME*. Typical installation directory for Tomcat on Windows is `C:/Program Files/Apache Software Foundation/Tomcat 5.5`.

Once Tomcat and Java 1.5 have been installed, you should download and install the QoS-Discovery component containing CoDIMS.

The QoS-Discovery for Tomcat (*qosdisc.zip*) is available at `http://codims.epfl.ch/` under the Download page. Unzip the latter file under `TOMCAT_HOME/webapps`; you will obtain the following structure: `TOMCAT_HOME/webapps/qosdisc`. Here's the structure of the package:

- `TOMCAT_HOME/qosdisc/WEB-INF/classes` contains the java classes, the embedded databases, the configuration files.

- TOMCAT_HOME/qosdisc/WEB-INF/classes/codims-home contains configuration parameters for CoDIMS, Catalog database, logs and scripts. We call this directory *CODIMS_HOME*.

- TOMCAT_HOME/qosdisc/WEB-INF/lib contains the jar files (especially *codimsd.jar*).

2.3.2 Prepare the Grid environment

In this section, we'll discuss how to prepare a distributed Grid environment so that you can parallelize the execution of the QoS-Discovery component over multiple nodes.

Our software uses the Globus Toolkit (GT4 Version 4.0.2) to build the Grid environment [3]. GT4 is a set of software components that implement Axis Web services for building distributed systems. The Web services are hosted by containers and we interact with them by starting the containers.

First, you have to choose a cluster of remote nodes (called the environment), i.e. choose a set of available machines (defined by their names or IP addresses). If you do not have an available cluster of nodes, you can simulate multiple nodes on your local machine. Each of these nodes, will hold a CoDIMS Web service (Globus Web service). Your local machine will also hold the Web service and will coordinate the communication between remote nodes. We will explain later how to install and call these Web services on each node.

We recommend using UNIX/Linux nodes for your remote cluster, as it is more convenient to call remote scripts for starting the Web services on these machines. There's no restriction for running Windows, UNIX or Linux on the local node.

Once you have chosen your cluster you are ready to install the Web services on each node of your cluster by following the installation steps below (on each node, including your local machine). Note that, if you do not have an available cluster of nodes and want to run multiple instances of Web services on your machine, you will only need to proceed with these steps once on your local machine:

- Download *codims.zip* from http://codims.epfl.ch/ under the Download page. This file contains: The Globus Java Core Source, the CoDIMS Web Service, The QoS-Discovery component jar file, the *CODIMS_HOME* directory containing some configuration informations.

- Unzip *codims.zip* in a directory of your choice.

- Set an environment variable called *GLOBUS_LOCATION* that points to that directory.

- If running UNIX or Linux, add *GLOBUS_LOCATION* directory and GLOBUS_LOCATION/bin directories to the path.

You have now created your remote environment and you will need to register it within CoDIMS. For doing that, edit the file CODIMS_HOME/codims.env under the Tomcat installation of your local machine only. The example above shows an environment (with *ID=1*) composed of two remote machines at EPFL and your local machine (holding the *LOCAL_WEB_SERVICE*):

- *ENVIRONMENT_ID=1*, you may want to choose a different id.

- *CODIMS_HOME = Path To CODIMS_HOME* on your local machine.
- *NODES=myMachine1.epfl.ch:8080;myMachine2.epfl.ch=8080.*
- *LOCAL_WEB_SERVICE = localhost:8080.*

On the other hand, if you want to test the parallelization on your local machine, you will edit *codims.env* as above. Note that, in this example, the two remote nodes are running on the localhost (on different port numbers):

- *ENVIRONMENT_ID=1.*
- *CODIMS_HOME = Path To CODIMS_HOME* on your local machine.
- *NODES=localhost:8081;localhost=8082.*
- *LOCAL_WEB_SERVICE = localhost:8080.*

You must commit now the modifications into the CoDIMS Catalog, which is the database storing all important configuration parameters. Go to `CODIMS_HOME/Scripts` under your Tomcat installation directory on your local machine and execute *codimsEnv.bat* if you are running Windows or *codimsEnv.sh* if running Unix/Linux.

2.3.3 Running the application

Now that you have downloaded and installed CoDIMS and Globus on each node, you are ready to start the Web services and run the application.

First, you have to start the Globus container on each remote node and on your local machine (otherwise CoDIMS Web Services won't be available). To start the container on a node, open a command window (or a Terminal), go to `GLOBUS_LOCATION/bin` and type the following: *globus-start-container -nosec -p portNumber*. The *portNumber* field is the port number on which CoDIMS Web Services will listen for incoming requests (previously defined in *codims.env*; and the *nosec* arguments indicates that we are running the Globus container without any security mechanism.

On Unix/Linux, you can run the script *container* from any directory to start the container. For stopping the container, identify the process id running on your machine and kill it (*kill* command).

We have also provided scripts for starting and stopping a list of Web services. See under `CODIMS_HOME/scripts/start.sh` and `CODIMS_HOME/scripts/stop.sh`. You can use these scripts to start Web Services running under UNIX or Linux machines. Note that, these scripts make use of SSH. So make sure that you have SSH installed on each machine or replace the command by RSH (which requires fewer configurations).

After starting the containers on each node of your environment (including the local Web service), you can run Tomcat on your local machine:

- Under Windows:
  - Run `TOMCAT_HOME/bin/tomcat.exe`.
  - Go to Windows Services (In Configuration Panel.Administrative tools) and start Tomcat.

- Under UNIX/Linux: run `TOMCAT_HOME/bin/startup.sh`.

By default, Tomcat runs on port 8080, you can change the port number by editing the file *server.xml* under `TOMCAT_HOME/conf` and replace the occurrence 8080 by your desired port number.

Finally, if Tomcat is started, go to `http://localhost:yourPortNumber/qosdisc/qosdisc.html` to test the QoS-Discovery component. You can run multiple discovery requests (one at a time). If you have finished with Tomcat or want to change your CoDIMS environment settings you should stop Tomcat as follows:

- Under Windows:
  - Close the Tomcat window.
  - Go to Windows Services (In Configuration Panel. Administrative tools) and stop Tomcat.
- Under UNIX/Linux: run `TOMCAT_HOME/bin/shutdown.sh`.

If you stop Tomcat and start it again, you will have to update you browser session (session identifier) by closing all open windows or by clicking on the *Refresh* button of your web browser.

2.4 Use CoDIMS from Java

In this section, we show how to install CoDIMS in order to run the QoS-Discovery process from Java.

2.4.1 Pre-requisites

First, download the source code (*codims_src.zip*) at `http://codims.epfl.ch/` under the Download page and unzip it in a folder (we call this folder *CODIMS_PROJECT*). Then use your favorite Java IDE to create a new project and import the files. Add all the jar files located in `CODIMS_PROJECT/lib` to your project. Afterwards, prepare your Grid environment; edit the file codims.env and run *codimsEnv.bat* (or *codimsEnv.sh* if you're using a Unix/Linux platform). Finally, start the containers on each node of your environment.

2.4.2 Running the application

Overview

The class QueryManagerImpl (in package *ch.epfl.codimsd.qeef*) defines an interface for communicating with CoDIMS:

- *QueryManagerImpl getQueryManagerImpl().*
- *RequestResult executeRequest(Request request).*
- *void shutdown().*

- *long executeAsync(Request request).*
- *ExecutionState getExecutionState(long requestId).*
- *RequestResult getRequestResult(long requestId).*

See the main method for a full example of utilization.

Prepare the request

In CoDIMS, a query demand is considered more generically as a request. Thus a user builds a request object and submits it to CoDIMS for execution. A request object comprises the following structure:

- The type of the request (it's ID). This one should be defined in the Catalog.
- A RequestParameter object (Package *ch.epfl.codimsd.query*) which encapsulates a HashMap object storing what the application needs to process. The user may want to add to the HashMap parameter values that the operators might need to access during the execution.
- Some tuning parameters added to the RequestParameter HashMap.

Initialize CoDIMS

Before, running the application we need to start CoDIMS by calling the method getQueryManagerImpl of the class QueryManagerImpl. Afterwards, the user may execute multiple queries without re-initializing.

Executing a request

There are two modes for starting a request execution in CoDIMS, single-user or multiple-user mode. In a single user mode, request execution blocks CODIMS until the latter finishes the evaluation. In this mode, a new request has to wait for the previous to end. In the multi-user mode, requests run asynchronously, freeing CoDIMS to accept and run new requests. In the multi-user mode, all common data-structures stored in the local node are shared between concurrent requests, optimizing the overall performance.

The single-user mode is obtained by invoking the *executeRequest* method of the class *QueryManagerImpl*. On the other hand, a call to the *executeAsync* method introduces a asynchronous evaluation of the requests, allowing concurrent query evaluations. When you call this method you get a request id that you can use afterwards to know the state of your execution and to get the final results. For doing this, you can periodically call *getExecutionState(yourID)* of class *QueryManagerImpl* and get an *ExecutionState* object encapsulating execution state information. If the *ExecutionState* indicates that the execution is finished (*isFinished()* method of class *ch.epfl.codimsd.qeef.ExecutionState*), you can use *getRequestResult* to retrieve the complete result.

Obtaining the request results

When the *ExecutionState* returns a *isFinished* indication, the execution has been successfully terminated and the results can be obtained from a *RequestResult* object. This object is composed of:

- *ResultSet* object containing a list of tuples. Each tuple is a single object containing the result (in the QoS-Discovery, this corresponds to a selected Web service and rank information).
- The *metadata* of the tuple, that helps you reading the object.
- The time of the execution.
- A result integer code. This parameter is not used in the QoS-Discovery request.

Finally, when you finish with CoDIMS you can close the system by calling the shutdown method.

2.5 More Detailed Documentation

More detailed documentation can be obtained from the CoDIMS Web Site [2] or directly from the CODIMS user manual [5].

3 Release Notes

3.1 Implemented Features of This Second Prototype

The following features are included in this second prototype release of our QoS-enabled service discovery component:

1. The component can select QoS-annotated Web services descriptions that match the user's QoS requirements and classify the result according to a ranking algorithm.

2. The QoS matching and ranking algorithms use information from the developed ontologies, utilizing the KAON2-based WSML reasoner as its underlying inference engine.

3. The ranking algorithm also uses the reputation-based estimation of various QoS parameters of each service to classify them (according to the user-defined criteria in the ranking algorithm).

4. The functionality-based service discovery component is also integrated into the discovery process.

5. All implemented discovery algebraic operators have been implemented and integrated into the QoS-enabled discovery framework: Bloom filter, Matchmaking, Ranking, and Reputation management operators. This enables the parallelization of the discovery process using the CoDISM-D query processing system to adapt with numerous number of service descriptions in the service repository.

6. The main class *ch.epfl.qosdisc.wsmx.QoSDiscovery* of our discovery component implements the standard *org.wsmo.execution.common.component.Discovery* interface specified by the latest release of the WSMX API.

7. The Derby DBMS is used to manage the database of the service descriptions and the QoS reports from the users.

This second (and final) prototype release also includes:

1. A list of developed ontologies tested successfully with the WSML-tools and reasoner. Among them are the QoS and ranking ontologies: upper ontologies and the derived ones for the following example application scenarios: the file hosting, hotel reservation, and stock-information broker from WP10.

2. A list of WSMO goals and Web service descriptions given the input from the use case partners.

3. An online demonstration enables the user to test all features of the QoS-enabled service discovery framework interactively using a dedicated Web-based GUI developed with Google Web Toolkit.

4. A Web page [1] with detailed documentation of the installation and using the component.

3.2 Known Issues

The following issues are left open at the time of writing this document:

- If a user would like to use the current functionality discovery component which is integrated in our QoS component, he or she should adhere to some restrictions which cannot be influenced by the QoS discovery component described in this document. The input WSMO service description and goal to be used should comprise only post-conditions and effects (and not preconditions or assumptions) in their capabilities. This is due to the fact that the current lightweight functionality discovery engine only considers the outcome of a service execution, and not the pre-state and post-state. Attentive readers can refer to the documentation page of the functionality discovery component at `http://wiki.wsmx.org/index.php?title=Discovery_Tutorial` for more detailed instructions.

- The current functionality discovery component we are using (dated August 09th 2006) still has some stability issues: the invocation of that component occasionally produces no effect. We are collaborating with the developers of the functional discovery component to help them identify the problem.

References

[1] *QoS-enabled Service Discovery Component: Main download page.* http://lsirpeople.epfl.ch/lhvu/download/qosdisc/, 2006.

[2] http://codims.epfl.ch.

[3] http://www.globus.org/toolkit.

[4] M. Hauswirth, F. Porto, and L.-H. Vu. *P2P and QoS-enabled service discovery specification. DIP Project Deliverable D4.17, available from* ***http://dip.semanticweb.org/documents/D4.17-Revised.pdf***, 2005.

[5] Othman Tajmouati and Fabio Porto. CODIMS User Manual. `http://codims.epfl.ch/download/codims.pdf`.