INVOKING ADAPTED WEB SERVICES USING A MULTI-AGENTS SYSTEM
The Framework PUMAS-AWS

Céline Lopez-Velasco, Angela Carrillo-Ramos, Jérôme Gensel,
Marlène Villanova-Oliver, Hervé Martin
LSR-IMAG Laboratory, SIGMA Team, B.P.72, 38402 Saint Martin d'Hères Cedex, France.

Keywords: Adaptation, Web Services, Web-based Information System, Agents.

Abstract: Nowadays, users access *Web-based Information Systems* (*WIS*) through various devices (desktops, laptops, *Mobile Devices* (*MD*) such as *PDA*). In nomadic environments, when a user queries a *WIS*, the result should be adapted according to her/his *context of use*. In our work, this context is represented by the personal characteristics of the user, the features of her/his access device and her/his location. In order to provide *WIS* designers with mechanisms for adapting information we propose *PUMAS-AWS*. This is a multi-agent framework which integrates the concept of *Adapted Web Services* (*AWS*, concept which introduces the adaptation into the *Web Services* standards) into the *PUMAS* framework (which is able to adapt, according to the *context of use*, the results of the queries of a nomadic user). This paper describes *PUMAS-AWS* which provides a nomadic user with adapted information retrieved from one or several *WIS* based on *AWS*, using two processes: the matching process from *AWS* architecture and the query routing process from *PUMAS*.

1 INTRODUCTION

Web Information Systems (*WIS*) are used for collecting, structuring and managing data which can be accessed though the Web. Nowadays, numerous kinds of devices (such as desktops, laptops, PDA) can be used for accessing these *WIS*. The challenge of *WIS* designers is to provide users with relevant information adapted to her/his *context of use*. According to (Dey *et al.*, 2000), a *context* is "*any information that can be used to characterize the situation of an entity*". In our work, the *context of use* (that we call *context* in the remainder of this paper) is composed of: the *technical feature*s of the access device (memory, screen size), the *personal characteristics* of the user (preferences) and the user's *location. This context* is used for adapting the content of the retrieved information. Adapting information according to user's characteristics is one of the main interests of the *Adaptive Hypermedia* community (Brusilovsky, 2001), but proposals made in this area generally concern standard configurations of the access devices (powerful and fixed devices). Some works focus on adaptation for *Mobile Devices* (*MDs*). For instance, *CC/PP* (by the *W3C*) describes device's features (e.g., screen size, memory) through metadata.. However, few address the issues of adaptation to the characteristics of the user. Our work aims at providing a user with adapted information according to her/his *context*. *XML* and *CC/PP* are used in our proposal in order to provide *WIS* designers with mechanisms for formalizing, defining and extending the definition of *context*.

In order to provide users with adapted information, architectures based on the agent technology have been proposed (Berhe *et al.*, 2004), (Gandon *et al.*, 2004). An *agent* is an autonomous and proactive entity which provides a user with services specified into a unified and integrated execution model (definition from *FIPA*). We have proposed a framework based on agents, called *PUMAS* (Carrillo *et al.*, 2005) in order to adapt information sent by one or several *sources of information* (*SoI*)*,* according to different criteria (e.g., user's profile, *MD* features). The *PUMAS* architecture is organized as a *Hybrid P2P* system (Shizuka *et al.*, 2004) where one agent is in charge of searching for information inside *SoI*.

Lopez-Velasco C., Carrillo-Ramos A., Gensel J., Villanova-Oliver M. and Martin H. (2006).
INVOKING ADAPTED WEB SERVICES USING A MULTI-AGENTS SYSTEM - The Framework PUMAS-AWS.
In *Proceedings of WEBIST 2006 - Second International Conference on Web Information Systems and Technologies - Internet Technology / Web Interface and Applications*, pages 127-132
DOI: 10.5220/0001255101270132

In this paper, we focus on *WIS* based on *Web Services* (*WS*). *WS* are technologies based on a standard architecture (*provider*, *registry* and *requestor*) and standard languages (*WSDL* and *SOAP* of the *W3C*, *UDDI* of the consortium *OASIS*) which allow interoperability between the entities of the *WS* architecture. The assumption made in this paper is that each query executed in a *WIS* is a call of a *WS.* Since the standards of *WS* do not consider adaptation, the notion of *Adapted Web Services* (*AWS*) has been defined in (Lopez-Velasco, 2005). This notion consists in invoking a *WS* which best satisfies the *context*. For this purpose, the *AWS* architecture is replicated on each *WIS* in order to select the best *AWS*. We show in this paper how *PUMAS* can be used in order to centralize the *AWS* architecture. Especially, the *Query Routing* process (Xu *et al.*, 1999) in *PUMAS-AWS* enables to find the "*best*" *AWS* (i.e. the one which is the most adapted to user's needs considering the *context*) among the set of *AWS* obtained from different *SoI* queried.

In this paper, first, we present the basis of this work (*PUMAS* and *AWS*). Then, we describe the *PUMAS-AWS* architecture. Finally, we present some related works, before we conclude.

2 PUMAS AND AWS

This section describes the basis of our proposal (*PUMAS* architecture and the concept of *AWS*), in order to explain the integration of this work.

PUMAS is a framework which offers a solution for retrieving adapted information (distributed between different *SoI*) through *MDs*. The architecture of *PUMAS* is composed of four *Multi-Agent Systems* (*MAS*, see Figure 1). First, the *Connection MAS* provides mechanisms for facilitating the connection from different kinds of devices to the system. Second, the *Communication MAS* ensures a transparent communication between the used access device and the system. It also adapts information according to the technical constraints of the user's device. Third, the *Information MAS* receives user queries, redirects them to the "right" *SoI* (e.g., the one which is the most likely to answer the query). This *MAS* adapts information according to the user's profile in the system (preferences) and returns filtered results to the *Communication MAS*. In charge of the adaptation, the agents of the *Adaptation MAS* communicate with the agents of the three other *MAS* in order to exchange information about the user (explicitly extracted from *XML* files or inferred from rules), and about the *context*. *PUMAS* allows to adapt information from different *SoI* whether they are located on servers or on *MDs*. In order to make it possible, one *agent* is executed on each *SoI*, and searches information inside it in order to answer user's requests. For a detailed description of *PUMAS*, see (Carrillo *et al.*, 2005).

The main advantage of the use of *WS* is that they allow remote applications to easily interoperate. The classical *WS* architecture relies on three standards (*WSDL*, *UDDI*, *SOAP*) and on three entities (*provider*, *registry*, *requestor*): the *provider* describes the *WS* with *WSDL* and publishes their descriptions in the *registry* (*UDDI*); The *requestor* researches in the *registry WS* which can meet to her/his needs. The communication between the *requestor* and the *provider* of the *WS* chosen is allowed by *SOAP*. However, in this architecture, no special attention is paid to the adaptation of information according to the *context*. In (Lopez-Velasco, 2005), the notion of *AWS* (*Adapted Web Services*) has been introduced as a classical and predetermined *WS* in order to be invoked in a specific *context*. Traditionally, *WSDL* describes a *WS* through its exchanged messages, proposed methods, data, structure and, communication protocol. An extension of *WSDL*, called *AWSDL* (*Adapted WSDL*), has been proposed (Lopez-Velasco, 2005). It allows the description of the adaptation proposed by the *AWS*, as well as user adaptation needs. An architecture, based on the classical one, has been created for the definition and management of *AWS*. This architecture extends the classical architecture with three modules: the *Filtering Registry* module, the *Filtering Profile* module, and the *Adaptation Matching Module*. The *Filtering Registry* module splits the *AWS* description provided by the provider into two parts: first the *WSDL* (classical description transmitted to the *registry*), second the *AWSDL* (description of the proposed adaptation recorded in a database). The *Filtering Profile* module splits the *requestor*'s query formulated in *AWSDL* into two parts: first the need of service (classical query transmitted to the *registry*), second the needs of adaptation (recorded in a database). The result of the classical query (a list of *WS*) is intercepted by the *Filtering Profile* module which transmits it to the *Filtering Registry* component. This component searches in the database the *AWSDL* description of the resulting *WS*. In order to perform the matching process, the *Adaptation Matching Module* collects the *AWSDL* description

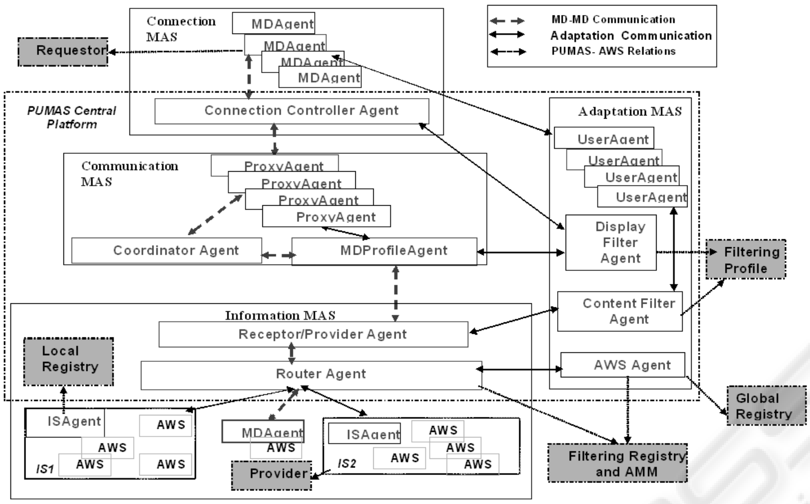

Figure 1: The *PUMAS-AWS* architecture.

of the *AWS* available and the query (in *AWSDL*). An *XML* file which contains the list of the *AWS* able to answer the user's needs of information and adaptation is sent to the *requestor*. The user chooses an *AWS* from the returned list and the communication between the *requestor* and the *provider* is established by *SOAP*.

We are interested in retrieving information from *AWS*-based *WIS* which use the *AWS* architecture. This architecture requires each *WIS* to have its own *Adaptation Matching Module*. in order to avoid this replication, we propose *PUMAS-AWS*, which integrates into *PUMAS* each entity of the *AWS* architecture. Through this integration, *PUMAS-AWS* manages the *Matching process* in a centralized way. Systems using *PUMAS-AWS* are able to call *AWS* available in one or several *WIS* (playing the role of the *provider*).These *WIS* are also considered like *SoI* (and become *registry)*.

3 PUMAS-AWS

In this section, we describe both the integration of each entity of the *AWS* architecture into *PUMAS* (see Figure 1) and three tasks: the *Management* of *AWS*, the *Matching Process* and the *Query Routing* process.

3.1 PUMAS Agents and AWS Entities

In *PUMAS-AWS*, the roles of some *PUMAS* agents change in order to integrate the activities of entities of the *AWS* architecture (*provider, registry, requestor, Filtering Profile, Filtering Registry,* and *AMM*). We also include a new agent which manages the *AWS*, called the *AWS Agent*.

The *Mobile Device Agent* (***MDAgent***) is executed on the user's access device. It is considered as a ***requestor*** since this agent sent queries for an *AWS* to the framework. The activities of the ***Filtering Registry*** module are performed by the *Router Agent* (***RA***) and the ***AWS Agent***. These agents search for the *AWS* which correspond to the user's needs.

The role of the ***Filtering Profile*** module is shared by the *Content Filter Agent* (***CFA***) and the *Display Filter Agent* (***DFA***), which manage the *context* defined by three *XML* files: the *Static Characteristics* (***SC***) file describing the permanent information of a user (e.g., *User ID*); the *Dynamic Characteristics* (***DC***) file which describes information about a user which varies during a session or between two sessions (e.g., location); and, the *Mobile Device Characteristics* (***MDC***) file which describes the features of the *MD* (e.g., screen size, memory). A file called *User's Preferences* (***UP***) describes the user's preferences, especially those concerning the criteria of adaptation (e.g., location). *PUMAS-AWS* manages these files as follows: the *SC* file is sent at the first connection of the described user to the *CFA* which consults it during the following connections. The *DC* file is sent by the *UserAgent* (which manage the user's profile) to the *CFA* at each connection in order to update information. The *DFA* manages the characteristics of a user's *MD* through the *MDC* file. It holds a *Knowledge Base* (*KB*) which contains information about features of different types of *MDs* (e.g., supported file formats) and acquired knowledge from previous connections (e.g., bandwidth for data transmissions). The *MDAgent* sends the *MDC* file to the *DFA*. The *DFA* updates the information about a *MD* during each session.

We distinguish two *registries*. First, the ***Local Registry*** is managed by the *ISAgent* ***(ISA)*** and contains the list of *AWS* provided by the *SoI* which hosts this agent. The *ISA* communicates with the ***AWS Agent*** in order to inform it about changes in the *AWS*. Second, the ***Global Registry*** is managed by the ***AWS Agent*** and centralizes the set of available *AWS*. The role of ***provider*** is played by the ***WIS*** (*SoI*). In each *WIS,* the *ISA* is in charge of notifying changes and updates of the *AWS* provided by the *WIS*.

Two agents perform the role of the ***AMM***: the *Router Agent* (***RA***) which redirects queries to the *right SoI* and applies a strategy which depends on adaptation criteria and the *AWS Agent* which manages the *AWS* (see following subsections).

3.2 Managing AWS

The *AWS Agent* (located into the *Adaptation MAS*) manages all information about *AWS*. This agent stores in its *KB* pieces of knowledge (called *facts* and described using the *JESS* syntax – http://herzberg.ca.sandia.gov/jess/) for each *SoI*. These descriptions of *SoI* are recorded in a *XML* file, named *WIS-characteristic* (***WISC***), sent to the *RA* by the *ISA*. The description of a *SoI* is composed of the *ISA*, the different *AWS* (*WSDL* and *AWSDL*) provided by *WIS*, the device where it is stored (e.g., server, *MD*) and its location.

The *WISC* file is translated by the *RA* into *JESS facts* (we use the primitive "*assert*" in order to define an instance of an *unordered fact* in *JESS* and store it into the *KB* of *JESS*). In this way, the *RA* knows the information of the *WIS* (its *ID,* its *ISA*, the information managed by this *WIS*, the location of the *WIS*, and the set of *AWS* provided). An example of a *WIS* which represents a *Travel Agency* is presented in the Figure 2.

```
(assert (WIS
(ID "TravelAWIS")
(ISAgent "TravelA-ISA")
(Managed-Information "Travels" "Holyday Stays"
"Flights" "Promotions")
(Device "server")
(Location "http://YourTravel.imag.fr")
(AWS-ID "AWS-T-1")))
```

Figure 2: An instance of the JESS Template which defines a *WIS*, playing the role of *SoI*.

This example defines a *WIS* identified by its name "*TravelAWIS*", and by its *ISA* "*TravelA-ISA*". This *WIS* manages information about *Travels*, *Holiday Stays*, *Flights* and *Promotions*. It is located on a server whose address is "*http://YourTravel.imag.fr*". This *WIS* hosts the *AWS* called *AWS-T-1.* There is one fact for the *WSDL* of an *AWS.* This description, which relies on the *WSDL* elements, is used to define an *AWS* (identified by an attribute *ID*): the traditional elements of *WSDL* (*import*, *types*, *message*, *portType*, *binding*, *service*) and, the *AWSDL* elements describing the level of adaptation proposed by the *AWS*.

3.3 Matching Process

The *Matching process* in *PUMAS-AWS* consists in retrieving the set of *AWS* which answer the user's query according to her/his *context*.

In order to illustrate the *Matching process*, let us consider a user, named Alex who has defined in his *User's Preference XML* file information about his preferences for his holidays. These preferences are interpreted by the *PUMAS-AWS* component as follows: Alex prefers the *holiday stays* at the *beach,* which have as departure city *New York* and he wants to travel with the *BestAirlines* company. In order to perform the matching, this process is split into two phases. First, *PUMAS-AWS* considers the user's needs in terms of the activities that he wants to execute on this system (e.g., to search information about holidays). According to the user's queries, the *XML* files and the *WSDL* description of the *AWS*, the *AWS Agent* selects a set of *AWS*. Second, this set of *AWS* is filtered by the *AWS Agent*. This *Filtering process* uses the criteria of adaptation (e.g., location). The *AWS Agent* searches for the *AWS* able to provide adapted information according to the four description files (*UP*, *SC, DC* and *MDC*). These files are compared with the *AWSDL* description of the *AWS*. The refined list of *AWS* is sent to the *RA* which launches the *Query Routing process* described in the next section. In our example, the *AWS Agent* selects among the *AWS* obtained in the previous phase, those which are the most adapted to Alex's preferences. After the *Filtering process*, the *AWS* selects the *AWS* which satisfy the preferences stored in the *UP* file (step considered in the *Query Router process*).

3.4 Query Routing Process

The *Router Agent* (***RA***) processes the *Query Routing* with the help of the *AWS Agent* which provides it with information about *SoI* and their *AWS*. In *PUMAS-AWS*, the *Query Routing* process has been simplified and adapted compared to the one performed in *PUMAS*. This process relies on two activities presented as follows.

The first activity of the *RA* is the *analysis of the query* which leads to a possible split of the query. This split is based on the one hand, on the criteria of adaptation (e.g., location, user's preferences), and on the other hand, on the knowledge (managed by the *AWS Agent* and the *RA*) about the *SoIs* and their *AWS* (see Section 3.2). The *RA* analyzes the complexity of the query with the help of the *AWS Agent*. The number of *AWS* to be invoked increases the work to be done by the *RA* in order to compile the results of queries (this task is not considered in this paper). After this analysis, the *RA* decides if it has to divide the query in sub-queries or not. For instance, the *RA* can split the Alex's query "*holiday stays at the beach, which leave from New York travelling through BestAirlines*" into several sub-queries if there is no *SoI* or an *AWS* able to answer the complete query. For example, in order to satisfy the Alex's query, it is necessary to consider the reservation of a flight, of a hotel and a car rental package. This query is split in the following sub-queries: (1) "*plane tickets* for *holiday stays* at the *beach*", (2) "*hotels* for *holiday stays* at the *beach*" and, (3) "*car rental packages* for *holiday stays* at the *beach"*; all of the queries consider *New York* as the departure city and *BestAirlines* as Airline Company. This split is done because several *SoI* (known by the *AWS Agent* and the *RA*) manage information.

The second activity is the *selection* of *AWS* answering user's queries. In order to answer a query, the *RA* invokes *AWS*. The *RA* asks the *AWS Agent* for *AWS* which can answer the user's queries. The *AWS Agent* performs the *Matching process* and sends this information to the *RA*. In order to select the *AWS* which match the best the adaptation need, the *RA* applies the adaptation criteria of the query (defined in the *XML* files).

In our example, let us suppose that three *AWS* satisfy the query "*plane tickets"* (*AWS-PT-1*, *AWS-AL1-2*, *AWS-AL2-2*), two *AWS* satisfy the query "*hotels"* (*AWS-H-2*, *AWS-CH-3*), and two *AWS* satisfy the query "*car rental"* (*AWS-RC-1*, *AWS-RC-4*). The facts received by the *RA* from the *AWS Agent* are shown in Figure 3. Let us suppose that Alex prefers *AWS* which are executed on servers located in agencies of the city where he works. The *RA* selects the following *AWS: "http://LocalAirline/services/AWS-AL2-2"*, and "*http://LocalHotel/services/AWS-H-2"* and "*http://RentalCar2/services/AWS-RC-4"*. The *RA* invokes these *AWS* and compiles the answers obtained from the different *AWS*, selecting the most relevant ones according to given criteria of adaptation of the user's queries before returning the result to the user.

```
; Facts representing the AWS which answer
the query "plane tickets"
(assert (AWS-for-query
(query "plane tickets")
(AWS-location
"http://YourTravel/services/AWS-PT-1"
"http://OneAirline/services/AWS-AL1-2"
"http://LocAirline/services/AWS-AL2-2")))
;Facts representing the AWS which answer
the query "hotels"
(assert (AWS-for-query (query "hotels")
(AWS-location
"http://LocalHotel/services/AWS-H-2"
"http://DestHotel/services/AWS-CH-3")))
;Facts representing the AWS which answer
the query "car rental"
(assert (AWS-for-query
(query "car rental")
(AWS-location
"http://CarRental1/services/AWS-CR-1"
"http://CarRental3/services/AWS-CR-4")))
```

Figure 3: Facts received by the *RA* from the *AWS Agent*.

4 RELATED WORK

A user would like the results of her/his queries to be adapted to her/his *context of use*. In order to achieve this, (Gandon *et al., 2004*) stress the need of considering knowledge about user's preferences and contextual features in order to search for information. Their approach is based on user's preferences, the rights granted to a user in order to change her/his preferences in a dynamic way and the representation of her/his contextual information. Unlike *PUMAS-AWS*, this work does not search information in *AWS*-based *WIS*. *PUMAS-AWS* uses a representation of the *context* which considers the user's characteristics as well as those of her/his *MD* and her/his location in order to adapt information. This *context* can be easily extended with characteristics defined in *XML* files and it is changed before, during and after the executions of user queries.

The work presented in (Berhe *et al., 2004*) proposes an architectural framework which exploits four profiles for adapting information to a user: *content or media* (format, size, location where media is stored), *user* (preferences), *device* (hardware and software capabilities), *network* and *service* (supported formats, network connection, bandwidth, latency). All of these characteristics can be defined in *PUMAS-AWS* in the *SC, DC* and *MDC* files. Moreover, these characteristics can be dynamically changed, unlike to the work of (Berhe *et al., 2004*).

This proposal unlike *PUMAS-AWS* does not consider information retrieval from different devices (servers and *MDs*).

Concerning *Web Service* (*WS*) adaptation, some works use technologies external to the classical *WS* architecture in order to perform different adaptations. For instance, (Pashtan *et al.*, 2004) propose to adapt the content delivered by the *WS* by transforming it using *XSLT* style sheets. However, their approach does not consider the user's context, considering only the user's device and preferences. (Keidl *et al*., 2004) propose an integration of the context definition into *SOAP* in order to find a *WS* able to satisfy user's needs considering her/his location, devices, presentation properties, and connectivity preferences. By using *SOAP* in order to discover a service, this proposal violates somehow the principles *WS* standard architecture, where *SOAP* is only used for communication purposes. Through *AWSDL*, our proposal respects the standard *WS* architecture.

5 CONCLUSION

In this paper, we have integrated the concept of *Adapted Web Services* (*AWS*) into the *Peer Ubiquitous Multi-Agent System* (*PUMAS*) framework. This result is *PUMAS-AWS*, a framework which focuses on the search of information among several *AWS-based WIS*. These *AWS* are able to answer queries in an adapted way, according to the *context of use* (composed of the personal characteristics of the user, the features of the access device, and the user's location). In this paper, we also highlight the roles of the *Router Agent* and the *AWS Agent* of *PUMAS-AWS* which are in charge of the *Matching* and the *Query Routing* process, fundamental pieces for adapting information. The *Matching process* consists in retrieving the set of *AWS* which answer the query according to the *context*. The *Query Routing* process relies on two activities: the *analysis of the query* and the *selection of AWS*. The first activity leads to a possible split of the query. The second one is achieved by the *Matching process*.

We now aim at improving the algorithms and mechanisms used in the *Matching Process*. We also need to define a system for establishing priorities among fixed criteria of adaptation in order to facilitate the process of selecting *AWS* performed by both the *AWS Agent* and the *RA*. In order to improve the matching process, we investigate semantic approaches, such as *WSMO* introduced by (Roman *et al.*, 2005). Finally, we are also interested in defining a mechanism for capturing the context in an automatic way.

REFERENCES

Carrillo-Ramos, A., Gensel, J., Villanova-Oliver, M., Martin, H., 2005. Adapted information retrieval in Web Information Systems using PUMAS. *In AOIS 2005, 7th Int. Workshop on Agent Oriented Information Systems*.

Lopez-Velasco C., 2005. AWSDL : une extension de WSDL pour des services Web adaptés. *In INFORSID 2005, 23th Conference INFORSID*, 133-148.

Berhe, G., Brunie, L., Pierson, J.M., 2004: Modeling Service-Based Multimedia Content Adaptation in Pervasive Computing. In *CF 2004, 1st Conf. on Computing Frontiers*. ACM Press, 60-69.

Brusilovsky P., 2001. Adaptive Hypermedia. In *User Modeling and User-Adapted Interaction*, 11 (1-2), 87-110.

Dey, A. K., and Abowd, G. D., 2000. Towards a better understanding of context and context-awareness. *In CHI'2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*.

Gandon, F., Sadeh, N., (2004, Oct). Semantic Web Technologies to Reconcile Privacy and Context Awareness. In *Journal of Web Semantics*, 1 (3).

Keidl M., Kemper A., 2004. Towards Context-Aware Adaptable Web Services. In *WWW 2004, 13th World Wide Web Conference*.

Pashtan A., Heusser A., Scheuermann P., 2004. Personal Service Areas for Mobile Web Applications. In *IEEE Internet Computing*, 8 (6), 34-39.

Roman D., Keller U., Lausen H., Bruijn, J., Lara R., Stollberg M., Polleres, A., Feier, c., Bussler, C., Fensel, D., 2005. Web Service Modeling Ontology. In Applied Ontology 1. IOS Press, 77-106.

Shizuka, M., Ma, J., Lee, J., Miyoshi, Y., Takata, K., 2004. A P2P Ubiquitous System for Testing Network Programs. In *EUC 2004, Embedded and Ubiquitous Computing*. LNCS, Vol. 3207, Springer, 1004-1013.

Xu, J., Lim, E., Ng, W.K., 1999. Cluster-Based Database Selection Techniques for Routing Bibliographic Queries. In *DEXA 99, 10th Int. Conf. on Database and Expert Systems Applications*. LNCS, Vol. 1677. Springer, 100-109.