WS-Advisor: A Task Memory for Service Composition Frameworks

Rosanna Bova*, Hye-young Paik†, Salima Hassas‡, Salima Benbernou*and Boualem Benatallah†
*LIRIS, University Lyon I, France
{rosanna.bova, salima.benbernou}@liris.cnrs.fr
†CSE, University of New South Wales, Australia
{hpaik, boualem}@cse.unsw.edu.au
‡LIESP, University Lyon I, France
hassas@bat710.univ-lyon1.fr

***Abstract*— With the proliferation of Web services, it is becoming increasingly important to support the users in selecting the most appropriate compositions of services for a task. We propose a new service discovery and selection framework that utilises the concept of task memories and a social network of task memories. A task memory captures the service composition history and their meta-data such as associated context and user rating. A network of task memories is formed to realise an effective task memory sharing platform among the users.**

I. Introduction

Today's foremost ICT challenge is to manage a large number of Web services over connectivity environments that are increasingly heterogeneous. What makes Web services paradigm very different from traditional distributed computing is that it promises to: adopt an environment in which users, services, and resources establish on-demand interactions; realise useful user experiences; and obtain services tailored to their time, place and their device capabilities. Typical activities of a service composition framework includes defining composite services, selecting actual service to execute a composite service, orchestrating service execution, and handling runtime exceptions. Advances in service oriented computing and semantic Web technologies provide foundations to enable automated services selection and aggregation [1].

A large body of research exists in the general area of Web services discovery, selection and composition. There have been important standardization efforts, initially focused on service description languages and interaction protocols, but they are now extended to other aspects including composition [2].

By leveraging the efforts in both Web services and semantic Web technologies, other services discovery and selection approaches emerged [4]. These approaches typically rely on description matching techniques (e.g., whether descriptions of services and requests are compatible). Descriptions refer to meta-data such as service capabilities and non-functional properties (e.g., quality of service properties) [1].

WS-Advisor [12] is an agent-based, service discovery, selection and composition framework. WS-Advisor extends existing work in the research areas and leverages knowledge about past experiences to improve the effectiveness of activities such as service selection, exception handling, and adaptation.

The design of the framework is based on the observations that most of the activities in service composition are repetitive in nature. That is, users tend to execute the same task over and over again, causing the service selection process to be repeated each time. If the service composition system can *remember* how the users reacted to the chosen services in the past, it can use the knowledge (e.g. the information about the contexts in which a certain combination services were considered most appropriate by users) to assist the future service selection process by making relevant recommendations automatically. The same idea can apply for handling exceptions. If the system learns how certain exception were dealt with in the past, the similarity between current and historical exceptions can be analysed to reuse, refine and generalise previous exception handling policies when possible. In [12], we introduced *task memories* as a way of capturing this knowledge.

In this paper, we focus on the issue of service selection in large-scaled and dynamic environments. We make the following contributions:

- We propose a social network of task memories as a platform for mass sharing of the knowledge acquired during service selection process and recommendations;
- We show how task memories can support various service selection and composition strategies, namely local, global and goal-driven service selection;
- We extend the architecture in [12] to introduce Social Network Agent and augmented Adviser Agent for sharing service recommendations.

II. WS-Advisor: Design Principles

The design of WS-Advisor leverages software agents, incremental knowledge acquisition, social networks in combination with ontologies as foundations to enable effective services selection:

- Agents are software entities that use their internal policies and knowledge to decide when to take actions that are needed to realize a specific goal. In our framework, internal policies can use context information (e.g., user preferences, device characteristics, and user location) to enable agents to adapt to different environments and user activities;

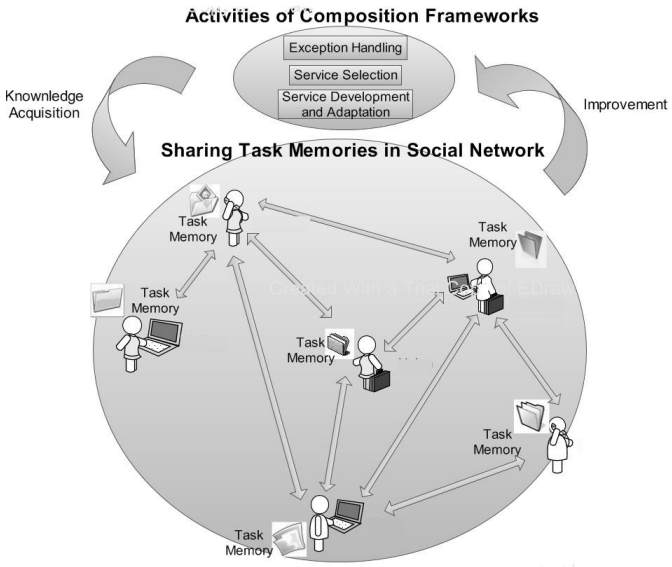

Fig. 1. Task Memories in Service Composition Framework

- Task memories represent the knowledge about services selection and contexts. This knowledge will be used to discriminate Web service offerings based on both functional and non-functional service properties (e.g., user preferences, contexts). WS- Advisor relies on incremental knowledge acquisition techniques where service selection policies are dynamically captured and continuously improved by analysing the ways in which they tend to use and rank services.
- Social networks have flourished with the advent of Web technologies to offer an information sharing infrastructure for individuals or groups. Our framework leverages the social network paradigm to extend the knowledge that an agent acting as recommender for a user (or a group of users) uses to select services that relevant for a given task. While each agent may accumulate its own knowledge based on its proper observations, sharing knowledge across agents allow these agents to leverage each other experiences in improving services selection recommendations. For example, the agent acting as a travel services recommender for a user A could reuse some service selection policies in an agent acting as a recommender for a user B, if A and B are part of a social network for sharing recommendations about travel services (e.g., airlines, hotels).
- Ontologies provide means for meaningful organisation of available resources (e.g., services, task memories) and information sharing among agents.

Agents extend services by embedding extensible knowledge and capabilities (e.g., context aware services selection policies) making them capable of providing personalised and adaptive service provisioning in dynamic environments. Incremental knowledge acquisition allows agents to continuously improve services selection policies. Social networks allow agents to share experiences. Ontologies support effective services discovery and knowledge sharing.

III. Concepts and Definitions

In this section, we briefly summarise the main concepts introduced in [12] to make the paper self-contained.

Service Ontology: The service ontology provides a description (e.g., domain, properties and capabilities) of potential services, In WS-Advisor, it is described by a set of *service categories* and each category is described by (i) a set of name-type-paired attributes that label properties of a service and (ii) a set of operations which characterises a service behaviour

Context Ontology: In our work, a context represents environmental or circumstantial factors relevant to effectively selecting services for a given task. We use a simple context ontology that consists of a set of context classes. Each class represents a specific aspect of task context (e.g., `Spatio-Temporal` context, `Computing Environment` context).

Task Definition: A task represents a set of coordinated *activities* that realize recurrent needs. For example, a business travel task may include activities such as hotel booking, car rental, flight reservation, meeting scheduling and attendee's notification.

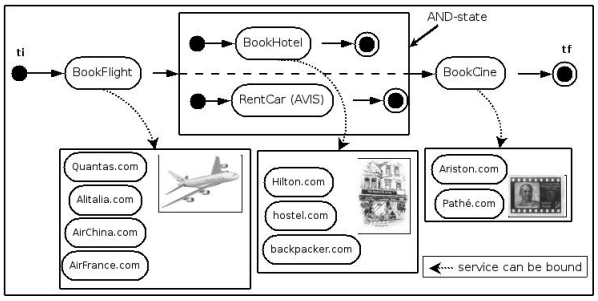

Fig. 2. Task Definition

A task is described in terms of services ontologies and is represented using state charts [8]. A state chart representing a user task consists of states and transitions. A state can be basic, or composite. Each basic state is labelled with an activity that refers to:

- *An operation of a concrete service*. The binding of an activity to a service operation is done at task definition time (e.g., RentCar in Fig. 2 is labelled with the actual service `Avis`).
- *An operation defined in a category of the service ontology*. The activity refers to an operation defined in a service ontology at definition time. The binding to a concrete service is done at run-time. In this case, an activity represents a request for a service instead of an invocation of a service (e.g., BookFlight and BookHotel in Fig. 2 represent a request for a service).

Annotating Tasks with Context: To cater for context-aware service selection, in addition to the activities and their dependencies, a task definition includes context attributes from the context ontology. The administrator associates each task

with its relevant contexts (e.g., for a travel booking task, the user's timezone, local currency, smoking preferences, type of Web browser, etc. may be relevant). Therefore, when a user chooses a task, any relevant contexts can be determined.

Context Summary Queries: A context summary query (CSQ) represents a context to be considered by the service selection process. It is specified using a conjunctive query of atomic comparisons involving context attributes, service attributes, service operation inputs/outputs, and constants. How the context summary queries are generated and associated with a task is explained in [12]. For example, for a flight booking task, relevant context attributes may be `origin`, `destination`, `price`, `seat class`, `currency`, etc. and a CSQ for a task could be expressed as a set of pairs `(c,v)` where `c` is an context attribute and `v` is the value (e.g., {(origin, "Sydney"),(destination,"Paris"),(seat class, "Economy")}).

IV. Task Memory

A *task memory* is a data structure which captures the information about the contexts and combinations of services that have been successfully used in the past to execute a given task. The service selection process considers task memories so that the selection is not only based on the description or content of services but also on how likely they will be relevant in a given context. To support different service selection approaches with task memories, we design different task memory schemas and the recommendation strategies. In the following, we introduce the different service selection approaches: namely, *local*, *global* and *goal-driven* and discuss the role of task memories in each.

A. Local Service Selection

In the local selection approach, the selection of the Web service for a given activity in a composite service specification is done at the last possible moment and without taking into account the other activities involved in the composite service. In other words, the service selection for a give activity in a composite service takes in consideration only the constraints expressed on the attributes of the activity, not on the combinations of activities. For instance, it is not possible to express the fact that the sum of the durations for two activities should not exceed a given threshold [1]. When this approach is used, we represent a task memory as a table that has two attributes, namely, context summary query, and individual service recommendations.

Recommendation strategy: For each context summary query, the task memory maintains the $K(K >= 0)$ most preferred services to execute for the given activity. Each service is associated with a positive weight value, called `Global Affinity` (`GA`), exceeding a predefined threshold (parameter sets by a system administrator, for example). Each table is associated with an activity and has three columns: `ID` (identifier of context summary query), `CQS` (Context Summary Query), `Service_GA` (services with their associated `GA`). For instance, the column `Service_GA` of Tab I shows examples of services and their associated `GA`. The global affinity of a service measures the relevance of the service in performing the activity in the given context in the task. More precisely, this value represents a weighted average of the values that measures the level of satisfaction of users, about the service, with respect to all the possible services that have been selected in that context. A more detailed description about the notion of global affinity and its computation is given in [9].

B. Global Services Selection

In this approach, the servive selection considers the constraints on the combinations of the all the activities involved in the task (i.e, enforce both local constraints and inter-activity constraints). The rationale behind this approach is that while, the local selection is much more simple from computation point of view, it may lead to optimised local selection, but the global quality of the execution may be sub-optimal [1]. For instance, if two activities are executed in parallel and need to syncronize upon completion, then it is not needed to optimise the duration of one task if it is known that the other task takes more time to finish. In this case, it may be worth optimising the price of the short activity and the duration of the long activity. Another example of a constraint that can be considered in global selection, is that the total price of the composite service execution should be at most 500 dollars.

In this case, a task memory is associated with a specific task and it captures the information about the contexts and the combinations of services that have been successfully used in the past to execute this task.

Recommendation strategy: For each context summary query, the task memory maintains the $K(K >= 0)$ most preferred combinations of services to execute a given task. Each combination of services is associated `GA`. A task memory is represented by a table with three columns. In this case, in the third column shows `Combination_GA` (combinations of services with their associated `GA`). Tab II illustrates some examples of combinations of services and their associated GAs.

C. Goal-driven Services Selection and Aggregation

This approach assumes that the service composition system receives a request and discovers both the services as well as the control flow among the selected services. While this is a far reaching objective, some progress has been in the areas of semantic web services [10] and AI planning [11]. They rely on machine understandable representations of service properties and capabilities; and reasoning mechanisms to select or aggregate services.

Our approach is independent of the implementation of the aggregation method. Whether the combination of services and their aggregation is computed automatically or semi-automatically does not affect how the the task memory is represented and constructed.

We assume that the system keeps track of the selected services and the way they were aggregated. We also assume that such aggregations are represented as execution path of a task as defined in [1]. To simplify, we represent an execution

TABLE I

AN EXAMPLE OF TASK MEMORY TABLE IN LOCAL SELECTION

ID	CSQ	Service_GA
CSQ1	origin = "Lyon" $\wedge$ destination = "Sydney" $\wedge$ (100 $<$ p $<$ 250)	{(Qantas, 0.6), (Alitalia, 0.4)}
CSQ2	origin = "Lyon" $\wedge$ destination = "Hong Kong" $\wedge$ (100 $<$ p $<$ 250)	{(Paradise, 0.7), (Hilton, 0.75)}

TABLE II

AN EXAMPLE OF TASK MEMORY TABLE IN GLOBAL SELECTION

ID	CSQ	Combination_GA
CSQ1	origin = "Lyon" $\wedge$ destination = "Sydney" $\wedge$ (100 $<$ p $<$ 250)	{[(Qantas, Hilton), 0.6], [(Qantas, Paradise), 0.4]}
CSQ2	origin = "Lyon" $\wedge$ destination = "Hong Kong" $\wedge$ (100 $<$ p $<$ 250)	{[(VolareWeb, Paradise), 0.7], [(Alitalia, Paradise), 0.75]}
CSQ3	origin = "Milan" $\wedge$ destination = "Sydney" $\wedge$ (300 $<$ p $<$ 500)	{[(Alitalia, Hilton), 0.8], [(Alitalia, Paradise), 0.65]}

path as a sequence of services $[s_1; s_2; ...; s_n]$, such that s_i is a service representing a node in the selected execution path. When this approach is in use, the task memory captures the information about the contexts and the paths of services that have been successfully used in the past.

Recommendation strategy: For each context summary query, the task memory maintains the $K(K >= 0)$ most preferred execution paths to execute a given task. Each execution path is associated with a `GA`, exceeding a predefined threshold. A task memory is represented by a table that as three columns: `ID` (identifier of context summary query), `CQS` (Context Summary Query), `ExecutionPath_GA` (a set of execution paths with their associated `GAs`).

V. WS-ADVISOR ARCHITECTURE

WS-Advisor aims to offer effective recommendations on "best-fit" services during the process of service selection. It has two core functions, namely: (i) constructing task memories and a social network of task memories, and (ii) using the task memories to make recommendations.

WS-Advisor is a multi-agent architecture which extends existing service matching and service selection techniques. They interact pro-actively to collect information for building the task memories and adapt continuously to provide effective service selection framework for service matching and selection.

In the following, we summarise each agent, namely: *user agent, adviser agent, social network agent, builder agent and context agent*. More detailed descriptions them are presented in [12]. In this section, we briefly overview these agents and discuss the design of the new social network agent and augmented adviser agent.

User Agent (UA). There are two types of users in WS-Advisor: administrators and end users. An administrator interacts with a user agent to manage tasks (i.e., create, update or delete). For an end user, a user agent acts like a proxy, through which the user can browse, select, and execute the tasks. When a task is chosen, the user agent performs the following automatic actions: (i) it contacts a context agent (see below) to retrieve context information, (ii) after obtaining the necessary contexts, it asks the adviser agent to recommend the services suitable for the chosen task. These recommendations are passed back to the user agent who makes the final decision on which services to run, (iii) when the services are finally chosen, the user agent interacts with a service orchestration engine (e.g., BPEL execution engine) to execute the task by invoking the involved services and orchestrating their interactions.

Adviser Agent (AA). The principal role of the advisor agent is to query the task memories for service recommendations to find the most appropriate combination of services for the user query and context.

A recommendation request from the user agent includes task attributes (e.g., departure date and destination city for a travel booking task) and context (e.g., current time and location) attributes. The knowledge that the adviser agent uses for recommendation is encoded in task memories.

The advisor agent maintains a *forwarding policy* that controls what should be done when recommendations are not found in its local task memories. The basic structure of a forwarding policy is as follows:

```
what theQuery|theCSQ
when empty|always|expand|busy
```

- **what**: This is used to decide what will be forwarded to the social network agent.
  - theQuery: the original user query containing both the task attributes and context is forwarded.
  - theCSQ: only the context summary query is forwarded.
- **when**: This is used to decide when the forwarding should be initiated
  - empty: forwarding occurs only when no recommendation could be made based on the local task memories.
  - expand: forwarding occurs even when some recommendations are found locally. This is used when, for example, the advisor agent wishes to expand the number of recommendations for better quality.
  - busy: forwarding occurs when the advisor is temporarily overloaded. The advisor has predefined value that indicates number of allowed incoming queries per time unit.
  - always: the query is always forwarded.

Social Network Agent (SNA). The social network agent maintains peer relationships with other social network agents,

effectively forming a network of WS-Advisors, the task memories. Currently, we assume that the initial relationships among social network agents are formed through invitations[1]. The social network agent then performs two main tasks: (i) building and maintaining the list of peers and their meta-data (e.g., description of the service/context ontology, task definitions and the summary of associated task memories, weights of the relationships indicating how "relevant" each peer is for a given task), (ii) forwarding a query by the advisor agent, collecting the results and relaying them to the advisor agent.

In a nutshell, this agent maintains *meta-data* and *task memory sharing policy*. The meta-data describes the information that are shared with peer agents. The task memory sharing policy specifies the allowed information dissemination and acquisition modes between the social network agent and its peers.

We consider various ways of describing the information sharing among agents in a social network. Let us denote a social network agent who is sending an invitation to another agent as `Source` and the recipient of the invitation as `Target`.

- Full Sharing: In this mode, the agents share their ontology, task definitions and context summaries. `Source` provides explicit mapping description specifying how the categories (respectively, the attributes/operations) in `Source`'s service and context ontology can be mapped to categories (respectively, the attributes/operations) in `Target`'s ontology. Therefore, when a query is forwarded from `Source` to `Target`, the query terms can be translated to the terms understood by `Target`.
- Category Sharing: In this mode, the agents share only the service and context categories in their ontology. The mapping only describes which category in `Source`'s ontology maps to which category in `Target`'s. The task definitions and context summaries can only be identified by to which categories in the ontology they belong. Therefore, when a query is forwarded, `Target` will use synonyms match to translate the query terms for the task and context attributes.
- No Mapping: When neither `Source` nor `Target` wishes to share the ontology, the description of the relationship carries no explicit mapping information. In this case, it is left to `Target` to figure out how to answer the queries forwarded by `Source`.

The basic structure of a task memory sharing policy is as follows:

- **whom**: This is used to decide to whom the query is forwarded.
  - **all**: all peers of the social network agent should selected for forwarding.
  - **random m**: maximum m number of peers are randomly selected.
  - `[p...]`: only the peers (p) that appear in the list are selected.
- **hop**: The hop limit is the number of subsequent forwarding that can occur once the forwarding has been initiated.

At the basic level of implementation, the social network agent can employ on-demand strategy for maintaining the list of peers, in which new peers are found only through receiving or sending invitations. However, more sophisticated form of the social network agent will try to maintain the list in a proactive way where the agent subscribes to a publication by other agents of certain content (e.g., a recommendation of service).

Builder Agent (BA). This agent is responsible for incremental knowledge acquisition in the task memories. It interacts with the user agent to gather service usage history (e.g., which services were recommended in which contexts, which of the recommended services were eventually chosen to be executed in the end, etc). It also continuously monitors and collects information about how the users rank the service performance after a task is completed

Context Agent (CA). The context agent collects an assortment of contexts from context providers and disseminates the information to the user agent. We assume that a context providing service, such as the one implemented in [7] exists and it will generate the context attributes and value pairs.

VI. Implementation Aspects

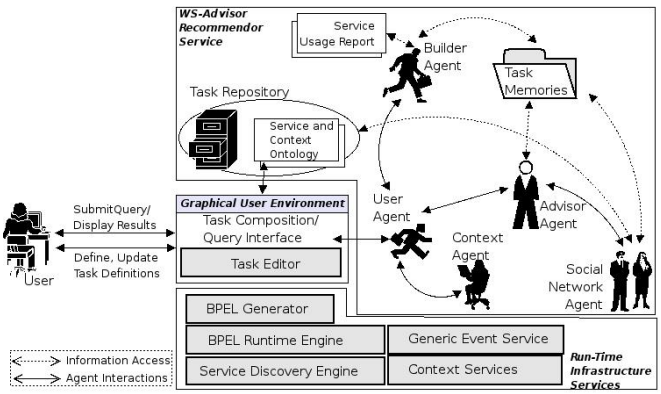

Fig. 3. Implementation Architecture of WS-Advisor

We adopt a layered architecture for the implementation of the whole WS-Advisor system. Figure 3 shows the elements of this architecture which are grouped into three layers: the agent layer (WS-Advisor recommender service layer); user layer; and the run-time infrastructure services layer.

a) The run-time infrastructure services layer.: This layer consists of generic services that we reuse from existing Web services environments to implement specific functionalities of the agents proposed in our approach. For example, the BPEL generator we reuse the model-driven BPEL generation tool from [5], [16]. For service discovery engine which facilitates the location of Web services from external service registries, we assume a service catalog search framework presented in [14]. The implementation of the context agent is a work in

[1]This is very much like the way Internet users are finding their "friends" in social networks.

progress and will rely on the context service implemented in the PCAP prototype [15] which is an extension of Self-Serv to cater for context awareness in service oriented architectures. The event monitoring service is used for tracking and monitoring service usage and relies on the event management component of the WS-CatalogNet prototype [14].

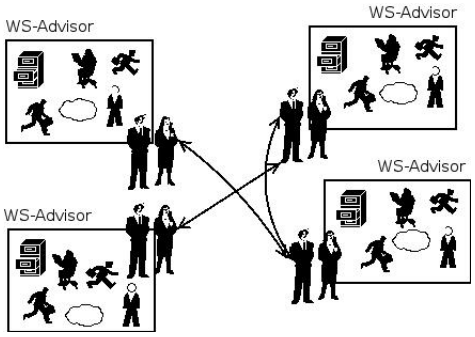

Fig. 4. Social Network Agents

b) The user layer.: This layer is the main layer with which the user interacts via the user agent. Also, through this layer, the user agent assists administrators in the creation and maintenance of tasks. This layer provides a task editor for describing a statechart of a task. Using the task editor, the users can also annotate the task definition with context and service ontologies. After the editing process, the user agent generates an XML file that represents a BPEL skeleton (a parametric process where invocations refer to service definitions instead of concrete services) and invokes a BPEL engine (ActiveBPEL[2]) to perform the execution of a task. The user agent provides means to inform the Builder agent about the selected services.

c) The WS-Advisor recommendation service layer.: It consists of services implementing the agents, mainly; the adviser, builder, social network and context agents. The implementation of the agents is based on Java, XML, and some generic services provider by the infrastructure layer of the architecture. We implement all agents Java classes and their behaviour (e.g., forwarding or sharing policies) are specified as XML documents. In particular, the adviser agent provides methods for querying a Task Memory which represented as an XML file. It also provides methods to query the service discovery engine. For the social network agent, we assume that an initial contact list consisting of names of other peers exist to form the network (Fig 4). The social network agent also provides a set of operation for creating, maintaining the list of peers and for forwarding queries to them. The implementation of these components relies on the services matching component of the WS-CatalogNet prototype [14]. The social network agent provides methods for querying peer task memories, subscription to recommendations on specific topics from peer agents, and notification of local recommendations to interested peer agents (i.e, agents which subscribed to the agent for recommendations on specific topics). The builder agent provides methods receiving notifications from the user agent, registering event patterns to the event monitoring service, and triggering actions for updating the task memory file. The context agent provides a method for querying context information.

[2]ActiveBPEL Engine, http://www.activebpel.org.

VII. Conclusion

WS-Advisor builds upon the building blocks of agents, social networks, and ontologies and provides a framework through which : (i) task memory, consisting of relevant services selection policies, can be incrementally constructed, (ii) social network relationships are leveraged to share task memories, and (iii) task memory is used to provide more personalized and context-aware selection of services. The proposed framework is an important step toward endowing services selection and composition techniques with capabilities to reuse leverage knowledge about past experiences. Ongoing work includes extending the architecture to support large-scale environments, and developing case studies to further study added value. Future work includes extending the notion of task memory to effectively support exception handling in composite services.

References

[1] Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Trans. Software Eng., Vol.30, (2004), pp.311-327.

[2] Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web services: Concepts, Architectures, and Applications. Springer Verlag (2003).

[3] Caverlee, J., Liu, L., Rocco, D.: Discovering and ranking web services with BASIL: a personalized approach with biased focus. ICSOC (2004), pp.153-162.

[4] Paolucci, M., Kawamura, T., Payne, T., R., Sycara, K., P.: Semantic Matching of Web Services Capabilities. ISWC (2002), pp.333-347.

[5] Sheng, Z., Benatallah, B., Dumas, M.: SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment, VLDB (2002) Hong Kong, pp.1051-1054

[6] Palau, J., Montaner, M., Lopez, B., de la Rosa, J.L.: Collaboration Analysis in Recommender Systems using Social Networks. CIA (2004), Erfurt, Germany, pp.137-151

[7] Dey, A. K.: Providing Architectural Support for Building Context-Aware Applications. PhD thesis, College of Computing, Georgia Institute of Technology, (2000).

[8] Harel, D., Naamad, A.: The STATEMATE Semantics of Statecharts. ACM Trans. Software Eng. and Methodology, vol. 5, no. 4, (1996), pp.293-333.

[9] Bova, R., Hassas, S., Benbernou, S.: An Immune System-Inspired Approach for Composite Web Service Reuse. Int. Workshop on Artificial Intelligence for Service Composition. Riva del Garda, Trento, Italy (2006)

[10] Bussler, C.: Semantic Web Services: The Future of Integration. ADBIS (2003), Dresden, Germany, pp.1-2

[11] Pistore, M., Barbon, F., Bertoli, P., Shaparau, D., Traverso, P.: Planning and Monitoring Web Service Composition. AIMSA (2004), Varna, Bulgaria, pp.106-115

[12] Bova, R., Paik, H., Hassas, S., Benbernou, S., Benatallah, B.:On Embedding Task Memory in Services Composition Frameworks. ICWE (2007), Como, Italy.

[13] Blake, M. B., Kahan, D., R., Nowlan, M., F.: Context-aware agents for user-oriented web services discovery and execution. Distributed and Parallel Databases 21(1) (2007), pp.39-58

[14] Paik, H.: Community-Based Integration Adaptation of Electronic Catalogs. PhD thesis, School of Computer Science, Univ. of New South Wales, Sydney, Australia. (2004).

[15] Sheng, Q.: Composite Web Services Provisioning in Dynamic Environments. PhD thesis, School of Computer Science, Univ. of New South Wales, Sydney, Australia. (2005).

[16] Baina K., Benatallah B., Casati F. and Toumani F.: Model-driven web service development, CAiSE (2004), Riga, Latvia, pp.290-306