On Embedding Task Memory in Services Composition Frameworks

Rosanna Bova[1], Hye-Young Paik[2], Salima Hassas[1], Salima Benbernou[1], Boualem Benatallah[2]

[1]LIRIS, University Lyon I, France,
{rosanna.bova, salima.hassas, salima.benbernou}@liris.cnrs.fr

[2]CSE, University of New South Wales, Australia,
{hpaik, boualem}@cse.unsw.edu.au

Abstract. With the increasing availability of Web services and adoption of services oriented paradigm, there is a growing need to dynamically compose services for realizing complex user tasks. While service composition is itself an important problem, a key issue is also how to support users in selecting the most appropriate compositions of services to fulfill a task. In existing dynamic services selection approaches, combinations of services are repeatedly discovered (e.g., using ontology-based matching techniques) and selected by users whenever needed. To improve their effectiveness, we propose a new technique that provides an efficient access to what is named a "task memory". A task memory is used to provide users with a context-aware service selection by recommending combinations of services that are most appropriate in a given context. A task memory is formed using the service composition history and their metadata. We present an incremental approach for building the task memory in which we monitor how users use and rank the services. The continuous updates of the task memory over time will result in more fine-tuned recommendations for composite services.

Keywords: composite web services reuse, context-aware composite web services selection, service oriented architecture.

1 Introduction

Advances in service oriented computing and semantic web technologies provide foundations to enable automated services selection and aggregation [1]. Coupled with other advances in communication technologies, these foundations constitute the pillars of a new computing paradigm in which users and services establish on-demand interactions, possibly in real-time, to realize useful experiences. This paradigm offers effective automation opportunities in a variety of application domains including personal information management, office tasks, travel, healthcare, and e-government. For example, a driver might use location, travel route computation, traffic information, and road conditions services to get timely information regarding a trip in progress. Business travelers can cope with schedule changes by seamlessly combining services to find and book hotels, search and book nearby rental cars, change flight reservations, modify meeting schedules and notify attendees [2].

A key issue to facilitate seamless and efficient composition of services is providing appropriate support for services selection. This is especially important in environments where there may be large number of services offering similar functionality [3]. Services discovery and composition are very active area of research and standardization. Efforts in these areas focused mainly on designing languages for process-based services composition (e.g., BPEL), designing rich and machine understandable representations of service properties, capabilities, and behavior, as well as reasoning mechanisms to select and aggregate services (e.g., OWL-S and services matching techniques) [4]. Main stream services discovery and selection approaches typically rely on descriptions 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) [3]. It should be noted that, in a description-based matching approach, identifying that a service has a capability to answer a user request, does not mean that the service will be selected by the user. For example, not all travel services offer airfares from Lyon to Sydney. Other approaches improve the effectiveness of description-based approaches by considering also content-based matching (e.g., using content summary [5] or using service probing [6]).

Although existing techniques have produced promising results that are certainly useful, more advanced techniques that cater for context (e.g., user location, computer environment), specially in large and dynamic environments, are necessary. This will relieve users from repeating the same selection refinement process to deal with a potentially large number of relevant services returned by a matching system every time they need to perform an activity. We observe that, while performing routine tasks, there is valuable knowledge being exposed to the service matching and selection component of a service infrastructure, that is, the information about the contexts in which a certain combination services were considered most appropriate by users. This information can be helpful in terms of reuse because users would select similar services in similar contexts (e.g., repetitive, regular tasks). Unfortunately, this information is not effectively captured and utilized in existing service matching and selection approaches. In this paper, we present an approach that leverages and seamlessly extends existing service matching and selecting techniques to cater for context-aware services selection by utilizing the knowledge on past experience. More precisely, we make the following contributions:

1. We introduce a notion of *task memories* that effectively represent the knowledge about service selection and contexts. We use task memories during services selection to suggest most relevant candidate services.
2. We use *incremental acquisition techniques to build and update task memory*. By applying continuous feedback and monitoring of ongoing usage of services, the system is able to maintain and evolve the task memory. Keeping the task memory up-to-date should result in more fine-tuned services selection.
3. We propose a multi-agent architecture; called, *WS-Advisor*, that seamlessly extends existing service matching and service selection techniques. The interactions among the agents are well coordinated to cater for a comprehensive service provisioning environment which supports effective capturing and utilization of user knowledge during service matching and selection.

2 WS-Advisor: Design Overview

The proposed architecture builds on existing services matching, selection, and composition frameworks. This is to take advantage of the already known techniques [7], [8], but, more importantly, to strengthened the notion of reuse in the frameworks. We propose that using incrementally acquired knowledge about service capabilities and their usage history during service matching and selection will help promote effective adoption of reuse. The added value of this extension is making the system (named WS-Advisor) capable of providing context-aware and adaptive service provisioning in dynamic environments.

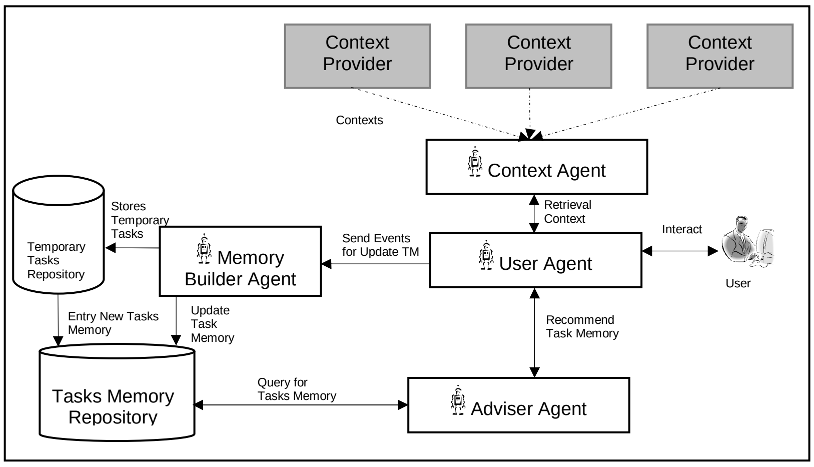

Fig. 1. Overview of WS-Advisor architecture

The main proposition of WS-Advisor is to offer effective recommendations on "best-fit" services during the process of service selection. The architecture as a whole relies on the notion of building, maintaining and querying a service usage history. By continuously monitoring which services are used in which context and how users rate the services after execution, WS-Advisor can build up extensible knowledge about a history of service usage. The knowledge is stored in the form of *task memories,* which are then queried during the selection of services to draw recommendations which are based on past experience (i.e., the best candidate services that performed well for the given task and context). In summary, WS-Advisor has two core

functions, namely: (i) constructing task memories, and (ii) using task memories to recommend "best-fit" services based on the past experience.
It is noted that WS-Advisor is based on a multi-agent architecture. Figure 1 shows the agents involved in the system, namely: *user agent*, *adviser agent*, *task memory builder agent* and *context agent*. These agents use their internal knowledge and policies to perform their functions. They interact pro actively to collect information for building the task memories and adapt continuously to provide effective service selection in dynamic environments. In the following, we introduce each 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, performing various actions on behalf of the user. A user agent maintains a folder of tasks (e.g., travel booking, organizing a board meeting). 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). 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. Briefly stated, a task memory consists of tuples, each tuple containing a combination of services, the contexts in which the services were selected and executed, a score indicating how "successful" the execution was. More detailed description of task memories and the scores will be given in the later sections.

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 and carry out updates in the task memories accordingly. We will discuss this agent in details in the later sections.

Context Agent (CA). The context agent collects an assortment of contexts from context providers and disseminates the information to the user agent. A context may refer to a user context (e.g., preferences, location, timezone), an environment context (e.g., hardware and software characteristics of the user's devices). We assume that a context providing service, such as the one implemented in [9], [10], exists and it will generate the context attributes and value pairs.

3 User Agent

In this section, we describe the concepts that are important, namely, *service and context ontologies; tasks*, to explain the activities performed by a user agent during task provisioning.

3.1 Concepts and Definitions

Service Ontology. Briefly stated, the service ontology provides a description (e.g., domain, properties and capabilities) of potential services that could be used to execute specific activities. A service ontology can be described using an ontology description language such as OWL-S. In our approach, the service ontology is described by a name that represents the domain of services and a set of *service categories*. A service category is specified by a set of attributes and a set of operations. An attribute describes a service property and is described by its name and type.

An operation describes a service behavior and is described by its name and signature (i.e, input and output parameters of the service). Categories within a service ontology can be related by specialization and generalization relationships. In this paper, we assume that service ontologies are available and accessible for instance from registries (e.g. UDDI registries). For example, in Figure 2, the domain `Travel` has a category `Transportation`, which is described using attributes `origin`, `destination` and `price`, etc., and this category has three sub-categories.

A service provider advertises a service by specifying which ontology the service is complaint to and the service categories that are supported by the service. Let us assume that the service `Alitalia` offers a range of flight information. The service may register itself with the `Transportation` ontology and advertise that it supports all operations in the category `Flight` as well as all attributes inherited from the categories `AirTransport` and `Transportation`.

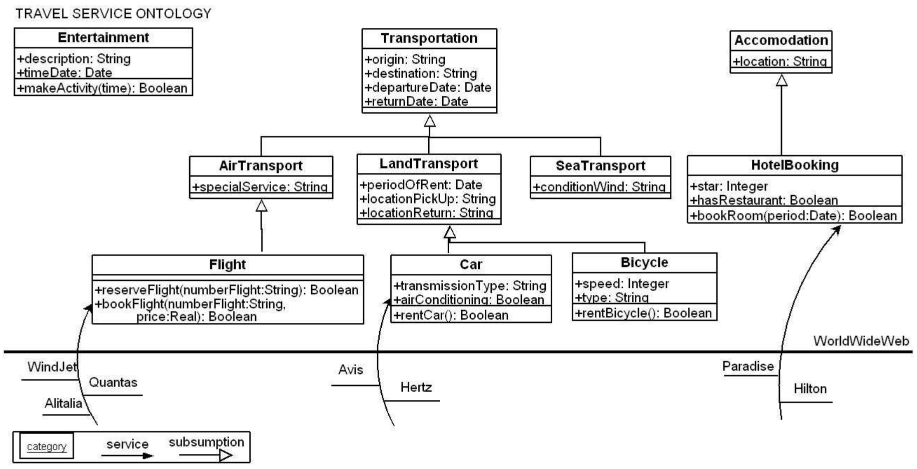

Fig. 2. An example of Service Ontology: “Travel Service Ontology”

Context Ontology. Various definitions exist in the literature for the notion context [11], [12]. For the purpose of our work, a context represents environmental or circumstantial factors that are relevant to effectively selecting services to perform 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, `ConditionalEnvironment` context, `User` context etc). These are generic classes in the sense that they are used to describe context of any task. Each class is described by a set of attributes representing specific state of the task environment. For instance, the class `Hardware` that is a sub-class of `ComputingEnvironment` contains the attributes: `memoryFree`, `cpuUsage`, `storage` and `network` and etc. It should be noted that, although the adopted context ontology has a limited number of context classes (for the sake of illustration), it is extensible: new classes can be added without fundamentally altering the service selection techniques built on top of this ontology.

Task Definition. A task in WS-Advisor represents a set of coordinated *activities* that realize recurrent needs. For example, a user may define a business travel task or a driving planning task. The business travel task may include activities such as hotel booking, car rental, flight reservation, meeting scheduling and attendee’s notification. A driving planning task may include activities such as gathering traffic and road conditions and producing an optimum driving route. An activity can be one of three types: (i) an *elementary* task that refers to an operation of an actual service, (ii) an elementary task that refers to an operation defined in a service ontology, or (iii) a sub-task (i.e., a task consists of other tasks).

The user agent provides support for defining new tasks and a repository for storing them. Tasks are, for example, defined by an administrator based on common patterns in recurring processes. A task is described in terms of services ontologies and is represented using state charts [7]. The choice of such notation is motivated by the fact that state charts offer main constructs that are needed to define typical user tasks such as sequence of activities, branching, and parallel activities. In addition, to their expressive power, well-defined semantics, state charts can also be translated to executable processes such BPEL. It should be noted however, that any other task modeling notation such as petri nets could be used to define tasks in our approach.

In a nutshell, a state chart representing a user task consists of states and transitions. A state can be basic, or composite. Each basic state is labeled with an activity that refers to:

– *An operation of a concrete service* in case the service is deemed relevant the corresponding activity whenever the task is performed. This means that, the binding of an activity to a service operation is done at task definition time.
– *An operation defined in a category of a service ontology*. This means that, the activity refers to an operation defined in a service ontology at definition time. The binding of this operation to an operation of a concrete service of the corresponding ontology is done at run-time. In this case, an activity represents a request for a service instead of an invocation of a service. Since, activities describing a user task are labeled with requests for services, concrete Web services belonging to the required service ontologies are selected during the execution of the composite task. Hence, it is possible to execute tasks in different ways by allocating different Web services to execute component activities in the task.

A composite state allows the nesting sub-tasks (represented as state-charts) inside a parent task. Transitions represent dependencies among the activities of a task (e.g., a transition may represent that an activity a1 should be executed after an activity a2 or a1 and a2 should be executed in parallel). A simplified state chart diagram specifying a "`Travel Planner`" task is depicted in figure 5. In this task, a search is performed to find a flight reservation service. After that, if the flight reservation is successful, an AND state follow, in which a search hotel booking service is performed in parallel with an invocation of a car rental service, and finally a search for an entertainment is performed. Note that activities `BookFlight` and `BookHotel` are labeled with requests for services whereas the activity `RentCar` is labeled with an invocation to an actual service (called "`Avis`"). The latter invocation style is useful when the service to use for executing specific task (e.g., preferred car rental service in the country of destination by the user of the task).

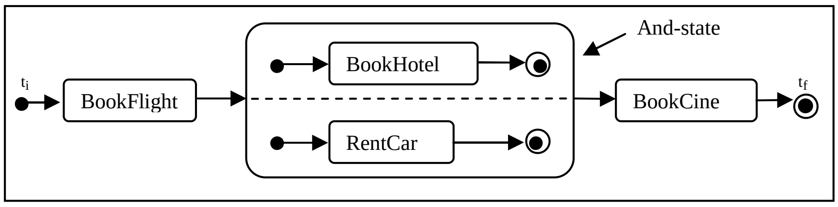

Fig. 3. State chart of the "Travel Planner"

Annotating Tasks with Context Information. To cater for context-aware service selection, in addition to the activities and their dependencies, a task definition includes context attributes from the context ontololgy. The administrator associates each task with its relevant contexts (e.g., for a travel booking task, the user's timezone, local currency, type of Web browser, may be relevant). Therefore, when a user chooses a task to perform, the user agent is able to determine the contexts associated with the task and contact the context agent to retrieve the values of each context attribute. For example, in the "`Travel Planner`" task, the administrator may choose the following relevant context attributes: for the activity `BookFlight`, the attributes `preferences` of the context class `User` and `time` and `location` from the context class `Spatio-Temporal`, because the user may have some preferences in the choice of airline company and this choice depends from time and location of this user; for the activity `HotelBooking`, the attribute `noise` from the context class `ConditionalEnvironment`, because the user may want, for example, a room with low noise level.

3.2 Provisioning Task

At a usage phase, a user chooses a task to perform from task repository (a repository maintained by the user agent). The user configures the required information to execute the task. In other words, user needs to specify a query that will be used by the system to select services to execute activities of the task. A user query is expressed in terms of attributes of service ontologies associated with the task. To simplify the process of expressing queries, each task is associated with task service schema (service schema for short). Given a task definition, a service schema describes the attributes that can be seen as a global schema for selecting the services to execute such tasks. The attributes of such service schema are derived from the attributes, inputs and outputs of operations referenced in that task definition. In addition, since our approach caters for task context, a query is expanded by the user agent to specify the current context. The user agent interacts with the context agent to get the values of context attributes that are relevant to a given task.

Example. Assume, that the schema of the task shown in Figure 3 contains: (i) the context attributes `preferences`, `time`, `location`, `temperature`, (ii) the service attributes `origin`, `destination`, `departureDate`, `returnDate`, `specialService`, `numberOfFlight`, `price`, `location`, `star`, `hasRestaurant`, `period`. The user agent retrieves the values of the context agent from the context agent. For instance, the result of querying of the context agent can be: (`preferences` = "Austrian Airlines", `time` = "8:30 AM CET", `location` = "Lyon", `noise` = "no", `temperature` = "25°C"). After that, the user fills the value of service attributes if desired and the input/output of operations. For instance the user query in this case can be: (`origin` = "Lyon", `destination` = "Sydney", `departureDate` = "02/01/2007", `returnDate` = "04/03/2007", `specialService` = "seat far to window", `numberOfFlight` = "OS 402", `price` "1000,00€", `location` = "Randwick", `star` = "2", hasRestaurant = "no", `period` = "03/01/2007 – 03/03/2007").

4 Adviser Agent

The core idea of our services selection approach is to *recommend combinations* of services that are most appropriate to meet the user's needs in given contexts. The recommendations are based on the past execution history of a task (i.e., task memories). In this section, we define the notion of task memories and discuss how the adviser agent makes service selection recommendations. The issue of building a task memory will be discussed in the next section when we discuss the memory builder agent.

4.1 Task Memory

A task memory is associated with a specific task and it captures the information about the contexts and combinations of services that have been successfully used in the past to execute the task. It is a kind of a dynamic folder that associates contexts to combinations of services. Dynamic, here, means that the contexts and the combinations of services may evolve over time. In this way, service 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. We represent a task memory as a table that has two attributes, namely, *context summary*, and *recommendations*.

Context Summary. Briefly stated, a context summary is a query representing a context that is considered by the system (or a system administrator) as relevant for selecting a combination of services to execute a task. It is specified using a conjunctive query of atomic comparisons involving context attributes, service attributes, service operation inputs/outputs, and constants. The second column of the table 2 shows examples of context summary queries. While context summaries could be defined using sophisticated query languages such as XQuery or Xpath, without loss of generality, we choose to use a simplified representation model in terms of attribute/value comparisons for clarity of presentation. The concept of context summary allows capturing a set of possibly relevant contexts to effectively select services instead of encoding all possible service selection queries which may incur high performance cost. In other word, the notion of context summary allows the adviser agent to maintain a partial, concise and effective index of service selection queries.

Table 1. An example of task memory table

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

We assume an administrator can identify a set of context summary queries that are relevant to a give task. This can be done by identifying a subset of attributes of the task schema that can be used to specify context summary queries. For each of these attributes, ranges of values are formed by dividing the domain of the attribute into a set of non-overlapping ranges known as Attribute Value Groups (AVGs). For nominal attributes, an AVG contains one or more distinct nominal values; for continuous attributes, an AVG specifies values range [5]. The union of AVGs of attribute is equivalent to the domain of the attribute. The Context Summary Queries (CSQs) are generated based on a cartesian product of these values. Table 2 lists examples of AVGs, assuming `origin`, `destination`, `price`, `star`, `memoryFree` and `temperature` are selected as summary attributes for the task "Travel Planner". The AVGs can be either manually defined by an administrator or discovered from query logs using query or context discovery techniques such as those presented in [13]. Once summary attributes are selected and AVGs are defined, the context summary queries are fixed.

Table 2. Example AVGs of summary attributes

Attribute	**AVGs**
Origin	'Lyon', 'Milan
Destination	'Sydney', 'Hong Kong'
Price	100 < p <250, 300 < p < 500
Star	1 < s < 3, 4 < s < 5
Temperature (°C)	20 < t < 25, 4 < t < 7

Recommendations. For each context summary query, the task memory maintains the K (K>= 0) most preferred combinations of services to execute a given task. Each services combination is associated with a positive weight value, called *Global Affinity* (GA), exceeding a predefined threshold (parameter sets by a system administrator, for example).A task memory is represented by a table that as three columns: ID (identifier of context summary query), CQS (Context Summary Query), Combination_GA (combinations of services with their associate Gas). For instance, the column Combination_GA of Table 1 shows examples of service combinations and their associated GAs. The global affinity of a services combination measures the relevance of this combination to perform a task in a given context. More precisely, this value represents a weighted average of the values that measure the level of satisfaction of users, about a services combination, which respect to all the possible combinations in the that have been selected in that context. A more detailed description the notion of global affinity and its computation is given in [14].

Making Services Selection Recommendation. During services selection, in a response to a query from the UA, the AA identifies the potential combinations of services having answers to the query. The AA provides an operation called `recommendCombinations(),` that takes as input a selection query and returns a set of service combinations that can be potentially used to executed the corresponding task. The AA matches the user query against context summary queries of the corresponding task memory. The matching process relies on *subsumption* (containment) or equivalence between a user query and context summaries queries. If no combination is found to be appropriate based on the task memory, the AA forwards the query to the matching agent to discover new possible combinations of services. For instance, the query Q: (category = "`TravelToSydney`", attributes: `origin`, `destination`, `price` and values: `origin` = "not defined", `destination` = Sydney, `price` = 150€) may not pass through the context filter of CSQ3 as the price is not included in its range. Hence any service associated to CQ2 is selected to answer to query Q, but CSQ1 can be used recommended to user.

5 Task Memory Builder Agent

The task of building a task memory is to associate context summary queries to combinations of services. This process is facilitated by a task memory builder agent (or simply builder agent). The Builder Agent (BA) is responsible for the incremental acquisition and to update of the elements the task memory table. Instead of asking an administrator to populate and update the task memory table, this agent incrementally captures the combinations of services that should be associated to context summary queries, by continuously monitoring how users use and rank services through interactions with the user agent.

This agent has access to operational knowledge such as service usage patterns as well as means for analyzing such patterns and updating task memories. There can be two approaches to build a task memory.

- A *lazy* approach consists to consider that the builder agent incrementally update a task memory starting from an initial table (e.g., an empty or a manually crafted table), during the service selection process. In this approach the builder agent maintains a usage table that consists of context summary queries and their associated service combinations. The usage table contains only combinations that have been used at least once with satisfaction. Every time, a combination is used with satisfaction (respectively dissatisfaction), the associated global affinity will be upgraded (respectively, degraded).
- An *eager* approach that consists to periodically search for services usage patterns by calculating the global affinities of previously selected service combinations. This can be achieved for instance by a logging facility associated with the builder agent. The agent logs events related to services selection. The logged information could be analyzed in real-time (during services selection phase) or periodically to identify patterns that help updating the task memory. Then, each pattern is associated to a task memory update operation (e.g., adding a new combination of services).

More specifically, in the current architecture, the builder agent relies on the following building blocks to incrementally construct selection policies:

- *Logging service selection events*. Table 3 summarizes basic events that are logged by the builder agent. Over time, these events are used as a basis to identify service usage patterns (e.g., identify that a combination of services needs to associate to a given context because the number of times this combination was selected with satisfaction is greater than a given threshold).

- *Task memory table update operations*. Table 4 summarized main task memory update operations. The evolution of a task memory table is realized through update operations.

- *Task memory table update rules*. Table 5 summarizes the main update operations supported in our framework. Operations to perform for updating a task memory table as a result of the occurrence certain service usage pattern are captured using *Pattern Action* where *Pattern* is a condition over service selection events, and *Action* is a table update operation. More precisely, a condition of a rule is a sequence over service selection events. A rule is defined for each update operation.

In the lazy update strategy, whenever a combination is selected, the builder agent checks if an update rule can be triggered (i.e, checks if the associated event pattern is true, and eventually performs the rule action if true). In the eager strategy, the agent relies on a pre-defined rule triggering policy (e.g, specified by an administrator). For instance a triggering policy may say "analyze the logged events periodically (e.g, each 2 days) to detect the occurrence of event patterns" or "whenever the task memory becomes a bottleneck." Note that a task memory might become a bottleneck when the AA forwards the user query to the matching agent frequently as the user is never happy with the recommendations of the AA or the agent does not find any relevant services combination.

Table 3. Selected events supported in WS-Advisor

Events	**Descriptions**
services_selected (cs, cqs)	The combination of services cs is selected by the AA as a relevant candidate in a context identified by context query summary cqs
services_used (cs, cqs)	The combination of services cs was selected by the AA as a relevant candidate and used with satisfaction by the user in context identified by context query summary cqs
services_discarded used (cs, cqs)	The combination of services cs is selected by the AA as a relevant candidate but discarded by the user in context identified by context query summary cqs

Table 4. Selected operations supported in WS-Advisor

Operations	Description
Upgrade_Score (cs, cqs)	The GA of combination cs is upgraded with regard to context query summary cqs
Downgrade_Score (cs, cqs)	The GA of the combination cs with regard to context query summary cqs
Add_combination (cs, cqs)	A new combination cs is added and its score is initialized with regard to context query summary cqs
Remove_Combination (cs, cqs)	A combination cs is removed with regard to context query summary cqs

Table 5. Selected rules supported in WS-Advisor

Rules	Pattern	Action
Upgrade_Score_Rule (cs, cqs)	<services_selected (cs, cqs), services_used (cs, cqs)>	Upgrade_Score (cs, cqs)
Downgrade_Score_Rule (cs, cqs)	<services_selected (cqs, tm), services_discarded (cqs, tm)>.	Downgrade_Score (cs, cqs)
Add_Combination (cs, cqs)	<services_selected (cqs, tm), services_used (cqs, tm)>	Add_combination (cs, cqs)
Remove_Combination (cs, cqs)	<services_selected (cqs, tm), services_discarded (cqs, tm)>	Remove_Combination (cs, cqs)

6 WS-Advisor: Implementation Architecture

We adopt a layered architecture for the implementation of the whole WS-Advisor system. Figure 4 shows the elements of this architecture which are grouped into two layers: the agents layer and the infrastructure services layer. The agent layer consists of services implementing the user, adviser, builder, 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. In other words, all the agents are implemented as Java classes. The infrastructure services layer consists of generic services that we reuse from existing Web services environments to implement specific functionalities of the agents proposed in our approach.

The user agent provides a GUI to assist administrators in the creation and maintenance of tasks. It provides an editor for describing a statechart of a task. The editing process consists of annotating states of a task with services descriptions based on services ontologies. In addition, a task is also associated to a number of context attributes from the Context ontology. 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). The implementation of the task editor and the generation of BPEL process skeleton rely on the state-charts editor and BPEL process generation components of the Self-Serv Prototype [7]. The user agent also provides a GUI to assist users in browsing tasks, selecting services, and execute tasks. It invokes the adviser agent to select services for executing a task. Once services are selected the user agent generates a BPEL executable process from the BPEL skeleton of the task and invokes a BPEL engine (ActiveBPEL) [15] to perform the execution of a task. The user agent provides means to inform the Builder agent about the selected services.

The adviser agent provides methods for querying a Task Memory which represented as XML file. It also provides methods to query the service discovery engine. The service discovery engine facilitates the location Web services from external service registries. The implementation of this component relies on the services matching component of the WS-CatalogNet prototype [8]. 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 event monitoring service is used for tracking and monitoring service usage and relies on the event management component of the WS-CatalogNet prototype. The context agent provides a method for querying context information. The implementation of this agent is a work in progress and will rely on the context service implemented in the PCAP prototype [7] which is an extension of Self-Serv to cater for context awareness in service oriented architectures.

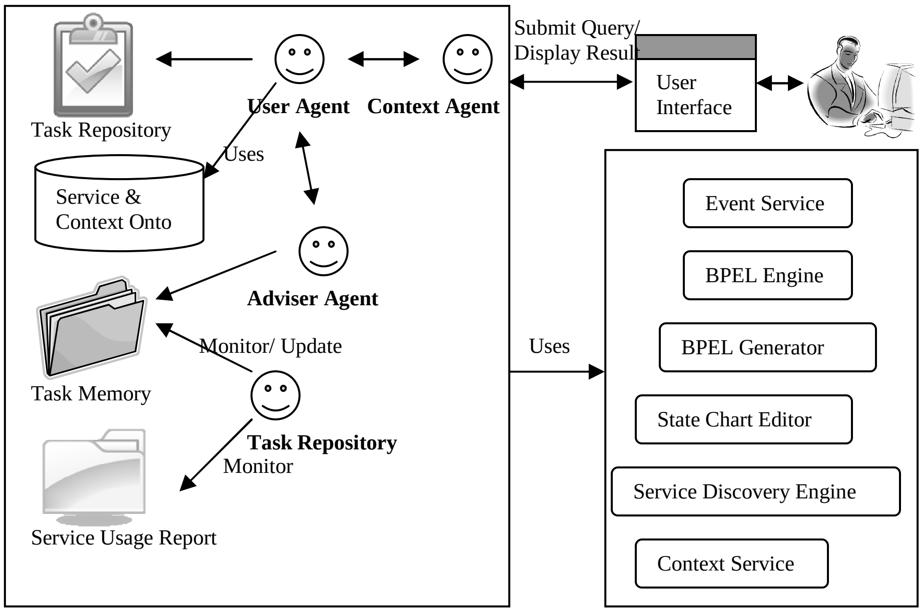

Fig. 4. WS-Advisor: Implementation Architecure

7 Discussion and Conclusions

A large body of research exists in the general area of web services discovery, selection, and composition. For example, early approaches based on the UDDI standard provide limited services search facilities, supporting only keyword-based search of businesses, services, category names, and service identifiers [16]. To cope with this limitation, other approaches based on semantic web technology, and in particular web ontology languages such OWL-S, to support service description and discovery emerged. Main stream approaches in this area focus on description-based matchmaking techniques based on subsumption and equivalence relationships [17]. As pointed out before, other approaches leverage content summarization techniques to improve the accuracy of description-based services selection and matching approaches. It should be also noted that the problem of description based matching has also been addressed by several other research communities, e.g., federated databases, information retrieval, software reuse systems and multi-agent communities. More details about these approaches and their applicability in the context of the semantic Web services area can be found in [18] and in [19].

Our work is also related to the general area of recommender systems, especially those based on multi-agents. (e.g., Amalthea [20], SAGE [21]). These efforts focused on analyzing documents (e.g., web pages, email folders) to recommend relevant documents as in search engines or products as e-commerce systems. Efforts in this area build upon personalization techniques in Web applications including content-based, collaborative and rule-based filtering [22]. Other agent-based approaches catered for context awareness in the orchestration of interactions among components of a composite service [23].

Our approach features embedding intelligence, consisting of task memories, into services composition frameworks allowing context-aware services selection. It builds upon services discovery and selection approaches to develop a context-aware services recommender facility during execution of routine tasks. This approach is based on the observation that, in performing routine tasks, the service matching and selection component of service infrastructure may produce valuable information on the contexts in which combinations of services where considered most appropriate by users. This information can be helpful to users in selecting services to perform a task because sometimes users would select similar services in similar contexts. Unfortunately, this information, which we call task memory in our framework, is not effectively captured in existing service matching and selection approaches.

We use task memory during services selection to suggest most relevant candidate services. We proposed to use incremental acquisition techniques to build and update task memory. A task is associated to an agent that monitors how users use and rank services. We believe that the proposed approach is an essential ingredient that will work in a tandem with services discovery and selection cooperative service techniques to provide more personalized and context-aware selection of services. Our future work will focus on experimenting with the proposed approach using some case studies to test its validity in real settings. We also plan to investigate the use of incremental knowledge acquisition techniques as means to learn context-aware selection policies. . In our future research, we plan to investigate adaptive CSQs such that the CSQs can be dynamically adjusted with respect to user queries.

References

1. Michael N. Huhns, Munindar P. Singh: Service-Oriented Computing: Key Concepts and Principles. IEEE Internet Computing, Vol. 9 (2005), 75-81
2. Teevan, J., Jones, W., Bederson, B.,B.: Special issue on Personal information management, Vol 49, (2006)
3. 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), 311-327
4. Medjahed, B., Bouguettaya, A.: A Dynamic Foundational Architecture for Semantic Web Services. Distributed and Parallel Databases, Vol. 17, (2005), 179-206
5. Sun, A., Benatallah, B., Hassan, M., Hacid, M. S.: Querying E-Catalogs Using Content Summaries. In Cooperative Information System. (2006)
6. Caverlee, J., Liu, L., Rocco, D.: Discovering and ranking web services with BASIL: a personalized approach with biased focus. ICSOC (2004), 153-162
7. Sheng, Z., Benatallah, B., Dumas, M., E. O.Y.: SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. In Proc. of the 28th International Conference on Very Large Databases. Hong Kong, China . September. (2002)
8. Baina, K., Benatallah, B., Paik, H., Rey, C., Toumani, F.: WS-CatalogNet: An Infrastructure for Creating, Peering, and Querying e-Catalog Communities. In Proc. of the 30th International Conference on Very Large Databases. Toronto, Canada. (2004)
9. Sheng, Q.: CompositeWeb Services Provisioning in Dynamic Environments. PhD thesis, School of Computer Science, University of New South Wales, Sydney, Australia. (2005)
10. Dey, A. K.: Providing Architectural Support for Building Context-Aware Applications. PhD thesis, College of Computing, Georgia Institute of Technology, (2000).
11. Lei, H.: Context Awareness: a Practitioner’s Perspective (invited paper) in IEEE International Workshop on Ubiquitous Data Management (UDM 2005), in conjunction with ICDE 2005, Tokyo, Japan, April, (2005).
12. Dey, A. K., Abowd, G. D.: Towards a Better Understanding of Context and Context-Awareness. Technical Report GIT-GVU-99-22, GVU Center, Georgia Institute of Technology, June (1999).
13. Chakrabarti, K., Chaudhuri, S., Hwang, W., S.: Automatic categorization of query results. In Proc. of ACM SIGMOD’04, Paris, France, June (2004), 755–766
14. Bova, R., Hassas, S., Benbernou, S.: An Immune System-Inspired Approach for Composite Web Service Reuse. Int. Workshop of ECAI06 Artificial Intelligence for Service Composition. Riva del Garda, Trento, Italy (2006)
15. ActiveBPEL Engine http://www.activebpel.org/
16. Dustdar, S., Treiber M.: A View Based Analysis on Web service Registries. Distributed and Parallel Databases, Springer, 2006.
17. Benatallah, B., Hacid, M., S., Leger, A., Rey, C., Toumani, F.: On automating Web services discovery. VLDB J. Vol.14, (2005), 84-96
18. Paolucci, M., Kawamura, T., Payne, T., R., Sycara, K., P.: Semantic Matching of Web Services Capabilities. International Semantic Web Conference, (2002), 333-347
19. Bernstein, A., Klein, M.: Towards High-Precision Service Retrieval. International Semantic Web Conference, (2002), 84-101
20. Moukas, A., Maes, P.: Amalthea: An Evolving Multi-Agent Information Filtering and Discovery System for the WWW. Journal of Autonomous Agents and Multi-Agent Systems, 1(1):59-88, 1998.
21. Blake, M. B., Kahan, D., R., Nowlan, M., F.: Context-aware agents for user-oriented web services discovery and execution. Distributed and Parallel Databases, Springer, (2006)
22. Paik, H.: Community-Based Integration Adaptation of Electronic Catalogs. PhD thesis, School of Computer Science, University of New South Wales, Sydney, Australia. (2004)
23. Maamar, Z., Mostefaoui, S., M., Yahyaoui, H.: Toward an Agent-Based and Context-Oriented Approach for Web Services Composition. IEEE Transactions on Knowledge and Data Engineering, (2005)