An Automatic Approach for Ontology-Driven Service Composition

Hua Xiao
School of Computing
Queen's University
Kingston, Ontario, Canada
huaxiao@cs.queensu.ca

Ying Zou, Ran Tang
Dept. of Electrical and Computer Engineering
Queen's University
Kingston, Ontario, Canada
{ying.zou, 8rt4}@queensu.ca

Joanna Ng, Leho Nigul
IBM Toronto Lab
Markham, Ontario, Canada
{jwng, lnigul}@ca.ibm.com

***Abstract*—Current service composition techniques and tools are mainly designed for use by Service Oriented Architecture (SOA) professionals to solve business problems. This focus on SOA professionals creates challenges for the non-expert users, with limited SOA knowledge, who try to integrate SOA solutions into their online experience. To shelter non-expert users from the complexity of service composition, we propose an approach which automatically composes a service on the fly to meet the situational needs of a user. We present a tag-based service description schema which allows non-expert users to easily understand the description of services and add their own descriptions using descriptive tags. Instead of specifying the detailed steps for composing a service, a non-expert user would specify the goal of their desired activities using a set of keywords then our approach can automatically identify the relevant services to achieve the goal at run-time. A prototype is developed as a proof of concept. We conduct a case study to compare the performance of our approach in automatic service composition with a baseline approach which consists of the manual process of searching for services using keywords. The case study shows that our approach can achieve higher precision and recall than the baseline approach.**

Keywords—tag-based service description; service discovery; service composition; ontology

I. INTRODUCTION

In today's on-line experience, an end-user, who is not familiar with Web services standards and tools, frequently re-visits Web sites and uses on-line services to perform repeated activities, such as on-line shopping. The end-user potentially composes an ad-hoc process to fulfill his or her needs. Such an ad-hoc process is characterized by a set of tasks performed by end-users without a strict execution order. For example, planning a trip is an ad-hoc process for many end-users. It involves several tasks, such as searching for flight tickets, booking a hotel, and checking the weather reports for the destination. These tasks can be performed in any order to achieve the goal of trip planning. More specifically, an end-user would manually browse different Web services to gather each piece of information to plan a trip. It is often challenging for end-users to compose the frequently used services as a process due to the detailed knowledge required for service composition.

In the current state of practice, developing Service Oriented Architecture (SOA) systems requires a large number of professionals (e.g., business analyst, system integrator and service developer) with strong SOA background. The development process involves various technical tools and languages to specify, compose, and deploy services. To produce a SOA system, the professionals in different roles and tools must interact in harmony. Unfortunately, non-expert end-users do not possess knowledge of most of these tools and lack the knowledge of SOA standards. In short, involving end-users in service composition has the following two challenges:

Complexity of service descriptions. The Web Service Description Language (WSDL) [8] is commonly used to define the programming interface of a service, such as the operations offered by a service and the format of messages sent and received between services. However, WSDL is too complex for non-expert end-users. WSDL is primarily intended for SOA experts to understand the interface and parameters of a Web service to correctly invoke a service.

Complexity of service integration. A system integrator can specify BPEL (Business Process Execution Language) [28] processes to compose Web services, using tools, such as WID (WebSphere Integration Developer) [15]. An ad-hoc process involves the dynamic integration of various services (e.g., Web services, and Web sites) on the fly. It is infeasible to expect an end-user to specify the details of each task and orchestrate a well-defined process in BPEL.

To help such end-users compose services for their daily activities, we propose an approach that hides the complexity of Web services standards and tools. Our approach automatically identifies services which reflect the situational needs of users. More specifically, we address the aforementioned challenges in the following two aspects:

1) To ease the end-user's difficulty in understanding the functional and non-functional properties of a service, we propose a service description schema that describes services using descriptive tags (i.e., keywords). The end-users can provide feedback based on their experience of using the services.
2) Instead of requiring end-users to specify the concrete tasks, the end-users only need to describe the goal that they want to achieve by invoking services using keywords. For example, the goal for planning a trip can be expressed using keywords, such as trip and travel. To derive the tasks for achieving the specified goal, we need to understand the semantic meaning of the specified goal. Ontologies capture the relevant information related to particular goals using expert knowledge. For example the ontology for the concept "travel" lists relevant concepts, such as "flight", "hotel reservation", and "weather". To have a better understanding of the specified goal, we search for existing ontologies that can expand the meaning of a specified goal. Furthermore, we

978-1-4244-5299-6/09/$26.00 (c)2009 IEEE

provide a technique that analyzes the identified ontology to automatically discover services and compose an ad-hoc process to achieve the specified goal.

The remainder of this paper is organized as follows. Section II introduces ontologies. Section III presents our approach to compose ad-hoc processes. Section IV describes a proof of concept prototype for our approach. Section V shows our case studies. Section VI gives an overview of the related work. Section VII concludes the paper and presents the future work.

II. MODELING AN ONTOLOGY DEFINITION

An ontology expresses common concepts (e.g., people, travel and weather), and the relations among these concepts. The semantic of a high-level concept is expanded into multiple more concrete sub-concepts. An ontology can be visualized as a graph that contains nodes representing a concept or a sub-concept, and edges representing relations between these concepts. Figure 1 illustrates an example ontology for defining the concept "travel".

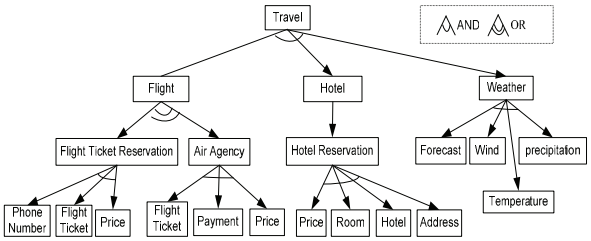

Figure 1. An Example Ontology Definition

Ontologies are manually specified using various standards, such as Web Ontology Language (OWL) [26] and Resource Definition Framework (RDF) [5]. Various ontology specifications define different types of relations among concepts/sub-concepts. To capture the general structure and concepts of ontologies specified in various languages, we create a general ontology graph that abstracts the relations specified in various standards into three types of edges. Figure 1 illustrates the various edges among concepts in an ontology graph.

- *subClassof:* a concept extends an abstract concept to convey more concrete knowledge. As shown in Figure 1, "*Fight*", "*Hotel*" and "*Weather*" are the subclasses of "*Travel*". The *subClassof* relation describes the parent and children relations among the connected concepts.
- *AND relation*: a concept is composed of its children. "*Weather*" is composed of "*Forecast*", "*Wind*", "*Temperature*" and "*Precision*".
- *OR relation*: two concepts are in an alternative relation. For example, "*hotel agency*" can be the alternative for "*hotel reservation*", meaning that a user can either book the hotel by themselves or asks a "*hotel agency*" to book it for them.

Essentially, the *subClassof* relation describes the extension of a parent concept to a set of children concepts. The *AND* and *OR* relations capture the relations among the child concepts. A terminal concept, such as *"Phone Number"* and *"Price"*, has no sub-concepts. A terminal concept delivers the most concrete information in an ontology graph. An internal concept, such as *"Air Agency"* and *"Weather"*, has at least one sub-concept. We also define the depth of a concept as the distance along the path from the root to the concept. For the example illustrated in Figure 1, concept *"Air Agency"* has a depth of 2, and the concept *"Payment"* has a depth of 3.

III. AN APPROACH FOR COMPOSING AD-HOC PROCESSES

An ad-hoc process records the tasks that need to be performed by an end-user. A task can be associated with more than one service of the similar functionally. For example, the task, "purchasing flight tickets", can be implemented by different travel agent services. In some cases, a few tasks have to be performed together. For example, two tasks, such as "selecting a book" and "providing payment information" must be performed before a book can be delivered. When more than one task must be completed, the relation among these tasks is treated as an "AND" relation. In the other cases, it is sufficient to conduct only one task among several tasks. For example, tasks, such as "buying a train ticket" and "buying a flight ticket" are alternative options for the transportation. Such an alternative relation among tasks is considered as an "OR" relation. An ad-hoc process contains a set of tasks and the "AND" and "OR" relations among tasks. The set of tasks can be performed in any order; hence this is an ad-hoc process. Figure 2 shows the schema for representing an ad-hoc process. The services fulfilling a task are either directly discovered from a service repository or composed by other ad-hoc processes (i.e., sub-processes).

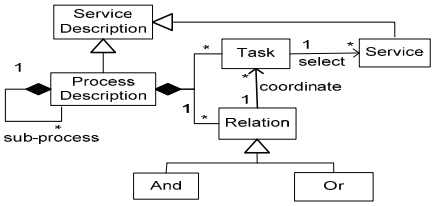

Figure 2. Description of an Ad-Hoc Process

To generate a meaningful ad-hoc process, it is critical to describe the service in an efficient way that allows end-users and composition tools to understand the properties of Web services. In Web 2.0, tags are a popular feature to describe Web resources. For example, Facebook [9] uses tags to depict images and seekda.com [23] takes tags to describe Web services. However, those tags are designed for the purpose of classification and searching. The tags are not organized in a structured way to ease the end-user to understand the detailed properties of services (e.g., the

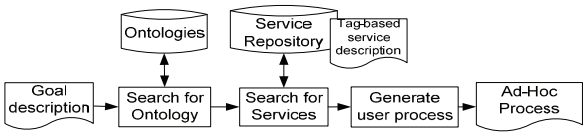

Figure 3. Steps for Composing Ad-Hoc Processes

operations of a service). Moreover, the tags can be redundant or irrelevant to a service. In our work, we propose a schema for associating Web services with structured descriptive tags. To ensure tags reflect the functionality of the services, the tags are initially extracted from WSDL. End-users can add new tags.

Figure 3 provides an overview of the steps for generating an ad-hoc process. To compose an ad-hoc process, a user simply describes a desired goal using keywords. We parse the identified ontology and represent it using the ontology graph discussed in Section II. In an ontology, the semantic of a high-level goal is expanded into more concrete concepts. We use the concepts as keywords to search for services in a service repository. To facilitate the reuse of the services when the same goal re-occurs, we abstract the discovered services into tasks and aggregate tasks into an ad-hoc process. The ad-hoc process can be stored and shared among multiple end-users. Finally, we display the generated process in a Mashup page so users can modify the process. In the following sub-sections, we discuss the steps in describing services, searching for ontologies to match with the goal description, and generating an ad-hoc process.

A. Tag-based Service Description

We propose a structured, tag-based service description schema which captures various properties of a service provided by both end-users and service providers to reflect the different perceptions of a service. Service providers define technical specification of the services. End-users can record perceived quality of the services and the delivered functionality. Figure 4 illustrates the schema for the tag-based service description. In general, we classify the tags into three categories:

General description is provided by the service provider. It specifies the basic characteristics of a service, such as version number, service name, and the URL address for accessing the service.

Functional description is provided by both service providers and end-users to describe the functionality of a service. A service provider publishes the name of a service, the operations and parameters as keywords. For example, an operation name, "*getWeather*", can be represented by a keyword, "*weather*". The constraints on using each operation are expressed as a set of self-defined tags and value pairs (i.e., weather forecast period = 7 days). Such functional description tags are automatically extracted from WSDL. Moreover, an end-user can add their own descriptions about each operation using a set of keywords.

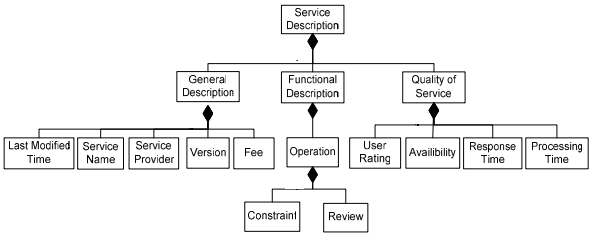

Figure 4. Schema for Tag-based Service Description

Many end-users may submit similar reviews. Similar to the indexing techniques used in existing search engines [6], we extract the meaningful keywords from the reviews and store them as tags.

Quality of services (QoS) specifies the quality attributes either perceived by end-users or measured by service providers. The availability of a service, response time and the processing time are major concerns when invoking a discovered service. The values for these quality attributes are monitored and provided by service providers. Furthermore, the end-users can submit their rating about a service. A set of tag and value pairs (i.e., availability = 99%, and user rating = excellent) are used to describe the quality attributes. More quality attributes can be added to extend the tag-based service description. The values for the quality attributes are used to select services when multiple services of equivalent functionality are returned.

The tag-based description for a service is represented in XML and stored in a service repository which links the WSDL for a service with the tag-based description. We design and develop tools to automatically extract meaningful tags from the user's input and WSDL files.

B. Searching for An Ontology for Describing User Specified Goals

An end-user describes a process as a goal using a collection of keywords (i.e., *keyword (G)={k_1, k_2, ..., k_n}, k_i refers to a keyword in the goal description*). For example, a travel planning goal can be represented as a set of keywords, i.e., *keyword(plan trip)={ travel, trip, hotel}*. An ontology can be defined to capture the expert knowledge of a user specified goal. Instead of predicting the possible users' goals and predefining the corresponding ontology, we search for relevant ontologies in the Web. This allows an end-user to specify any goals for the ah-hoc processes. To improve the chances for discovering the ontology, we also collect the set of synonyms of the keywords used in the goal description (i.e., *keyword (G)*). As an optional choice, we allow users to specify the tasks that they want to perform using keywords. For example, a "planning travel" goal contains tasks, such as car rental, hotel reservation, and transportation. Similarly, we collect the synonyms for the task descriptions. We denote the task descriptions as *keyword (T) = {tk_1, tk_2, ..., tk_m}* where tk_i refers to the keyword for describing a task.

An ontology of the specified goal is identified when the root of an ontology matches one of the keywords in *keyword (G)* (i.e., the set of keywords for goal description). For example shown in Figure 1, when the root concept, "travel" is matched with a "Travel to New York" goal description, the ontology is returned. If a goal description is matched with a concept defined within an ontology instead of the root concept, we retrieve the matched concept and its sub-concepts (e.g., shown in Figure 1, the concept "hotel" and all the related sub-concepts, such as "hotel reservation" and "price"). In some cases, more than one ontology can be matched with the goal description. To select an appropriate one, we count the frequency of the keywords in the set of the task description (i.e., *keyword (T)*) and the goal description (i.e., *keyword (G)*) appearing in each ontology. We select the ontology with the highest frequency of the provided keywords. A user can also inspect the ontologies and manually select one.

C. Searching for Services

The identified ontology can provide more detailed description about a user's goal. Essentially, the concepts defined in ontology capture the characteristics of the functional requirements for desired services that help achieve a user's goal. We use concepts as criteria to search for the matching services. More specifically, we group the names of a concept and its sub-concepts as a set of keywords, i.e., $concept(c_0) = \{c_0\} \cup \{c_1, c_2, c_3, ..., c_m\}$. c_0 is the name of a concept in an ontology. c_i refers to a sub-concept expanded from c_0. For example, the "hotel reservation" concept is expanded into a set of detailed sub-concepts, denoted as a set of concepts, i.e., *concept (hotel reservation) = {hotel reservation} ∪ {price, room, hotel}*. Each concept c_i has its own set of synonyms, i.e., $syn(c_i)=\{s_{i1}, s_{i2}, ..., s_{in}\}$. For example, the concept, "*hotel reservation*", has a set of synonyms, such as room booking (i.e., *syn(hotel reservation)={room booking}*). To retrieve relevant Web services from a service repository, we combine the concept set and synonym sets into a keyword set (i.e., $con\_keys(c_0) = concept(c_0) \cup (\bigcup_{i=0}^{m} syn(c_i))$) to search for the matching services in a service repository.

$$\text{SIM} = \frac{\text{\# of matched keywords}}{|\text{n}|} \quad (1)$$

n is the total number of tags in general description and functional description for a service

As discussed in Section III.A, each service, s_i, in a repository is described by a set of tags that specify the general information and the functionality. Therefore, we form a set of keywords for describing a service using tags, i.e., $ws\text{-}keys(s_j) = \{t_1, t_2, ..., t_z\}$, where s_j is a service and t_i is a tag of a service. To discover a service, we count the number of matched keywords between the concept description keywords (i.e., $con\text{-}keys(c_0)$) in the ontology and service description tags, i.e., $ws\text{-}keys(s_j)$) in the service repository. The similarity degree of the concepts and services are defined in equation (1). The similarity degree ranges from 0 to 1. A higher value means that more tags in a service description are matched with the supplied concepts. A high value indicates a high degree of similarity between the concepts and the services.

As a result of service discovery, we locate the services with the required functionality. When many services are matched, we can sort services according to the similarity degree from high to low. When two services have the same similarity degree, they are sorted using the values of QoS description provided in the tag-value pairs specified in the service description. For example, the discovered services are sorted using the values of the processing time given that the discovered services have the same similarity degree. An end-user needs to interpret if the high value of a quality attribute is more desired for the returned services.

D. Generating Ad-Hoc Processes

The entire set of concepts defined in an ontology graph could be used to search for all possible services. However, a large number of services could be returned without any logical relations. Returned services may be redundant. It is a tedious job for end-users to manually select desired services. We develop a technique to organize the returned services in a logical structure (i.e., ad-hoc process) by: 1) sorting the functionally similar services under one task; and 2) inferring the relations of the corresponding tasks. Essentially, ad-hoc processes provide abstract description on a list of unique tasks that a user must perform in order to achieve the goal. More specifically, we generate an ad-hoc process in two steps: 1) analyze the concepts in the ontology graph to identify a list of unique tasks which are associated with one or more functionally similar returned services; and 2) identify the relations among the tasks.

To identify tasks, we first decompose the ontology graph to locate a subset of concepts used for service discovery. To provide more concrete information in the service discovery, we use a depth first traversal to find the most concrete concepts in the search criteria. In particular, the terminal concepts represent the most detailed knowledge about the root concept which is the goal specified by users. Therefore, we visit the terminal concept in the furthest apart from the root concept. However, simply using a terminal concept in the search criteria may also prevent us from discovering some services since a single keyword from the terminal concept provides limited knowledge. Similar to the most of search engines, which use the expanded query to search for the relevant documents [1] [10], we extend the search keywords by including the parent of the terminal concept of the longest path, and the sibling terminal concepts as the keywords to search for a service in a service repository, as discussed in Section III.C. For example shown in Figure 5, *"Phone Number"* is a terminal concept in the longest paths. Its parent, *"Air Agency"* and the terminal concepts, *"Flight Ticket"* and *"Price"*, are provided as keywords to search for services (i.e., con-*keys(Air Agency)={Air Agency, Phone*

Number, Flight Ticket, Price}). Once a service is identified from the group of concepts, these concepts are marked as a task, shown in Figure 5. We derive the task name from the name of the parent concept. For the example shown in Figure 5, the task corresponding to concepts *"Phone number"*, *"Flight Ticket"*, *"Price"* and *"Air Agency"* is named after the parent concept, *"Air Agency"*. If no relevant Web services are retrieved from the path, we remove the failed concepts (i.e., the parent along the children) from the ontology graph. We recursively identify the next terminal concept with the longest path and repeat the same procedure, until all the terminal concepts are visited.

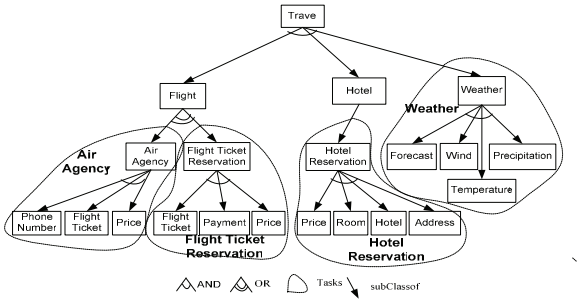

Figure 5. An Example of an Ontology Graph

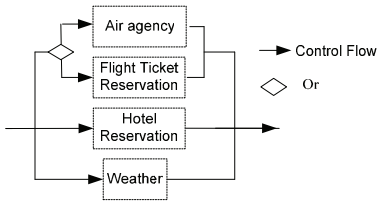

Figure 6. An Example Ad-Hoc Process

We identify the relations (i.e., AND, OR relations) among tasks by inheriting the relations from the closest common concept of the tasks. For the example shown in Figure 5, tasks *"Air Agency"* and *"Flight Ticket Reservation"* have the common concept, *"Flight"*. The basic control structures among tasks are determined by the edges emanating from the closest common ancestor of the two tasks. For example shown in Figure 5, the control structures of tasks, *"Air Agency"* and *"Flight Ticket Reservation"* are determined by the edges of the *OR* relation emanating from *"Flight"*. Similarly, tasks *"Hotel Reservation"* and *"Weather"* are in an *AND* relation. We traverse the ontology graph from the tasks to the root concept to recognize the hierarchal structure among control flow relations. An *OR* relation in an ontology graph is interpreted as an alternative control flow among two tasks. An *AND* relation in an ontology graph is converted into a parallel control flow which describes a set of tasks running in any order. As a result, Figure 6 shows the generated ad-hoc process from the ontology graph depicted in Figure 5.

IV. IMPLEMENTATION

We built a prototype to help end-users to generate ad-hoc processes by specifying a goal. We use the IBM WebSphere Service Registry and Repository (WSRR) [16] to register and manage Web services. To display the interfaces of selected services and invoke services, we use the IBM Mashup Center [14] as a service Mashup platform to integrate various Web services.

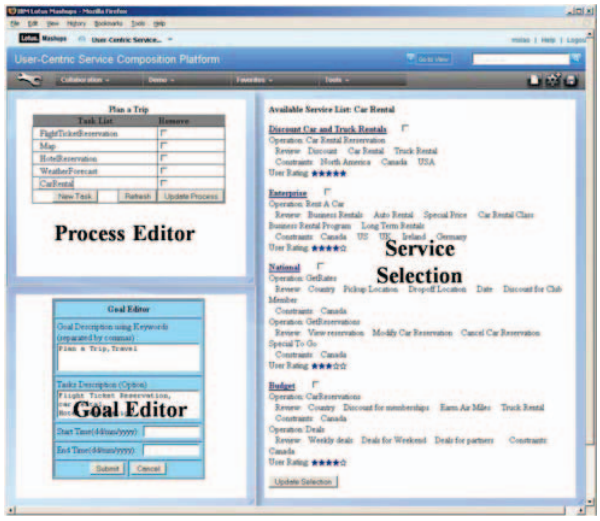

Figure 7. Annotated Screenshot for Our Prototype

Figure 7 is an annotated screenshot of our prototype. A user can specify their goal (e.g., plan a trip to New York) in the *Goal Editor*. In our current implementation of the prototype, the ontologies are manually searched using Swoogle [27], a search engine for ontologies, and imported into our prototype to ease the analysis. An ad-hoc process is automatically generated to capture a set of tasks that meet the specified goal shown in the *Process Editor*. A task in a generated ad-hoc process can be associated with one or more services. As shown in Figure 7, once a user selects the "*Car Rental*" task in the *Process Editor*, the associated services are automatically displayed in the *Service Selection Panel* on the right side of the markup page. We allow a user to select the most desirable services.

A user can refine and customize the ad-hoc process in the process editor. A user can remove a task if it is not needed by selecting the "*Remove*" check box. A user can also add a new task by specifying keywords for searching for services. We record the modifications as the user's preferences. When a user specifies the same goal, our prototype provides the previously refined ad-hoc process.

V. CASE STUDIES

The objective of our case study is to evaluate the effectiveness of our approach that eases end-users to automatically compose services without the knowledge of SOA technology. We want to examine 1) if our approach

can effectively generate ad-hoc processes using ontologies; and 2) if the tag-based description helps to locate the relevance services with high precision and recall.

We manually registered 504 Web services into our service repository. In addition, we searched for the related ontologies and imported them into our ontology database. Each of the services is described using the proposed tag-based service description.

A. Evaluation of the Generation of Ad-Hoc Processes

We recruit a Master's student who has no knowledge of SOA, as an end-user. We gave the subject a 15-minutes tutorial on how to use our prototype and asked him to compose ad-hoc processes by specifying five different goals. Five processes are automatically generated from five ontologies (listed in Table I) found in the Web to achieve the five goals respectively. The characteristics of the generated ad-hoc processes and the corresponding ontologies are specified in Table I. For example listed in Table I, Process 1 is generated from the "*Travel*" ontology which contains 44 concepts. We identify 5 tasks from the "*Travel*" ontology listed in Figure 7. As shown in Figure 7, task "*Flight Ticket Reservation*" is derived from concepts, such as "*Airport*", "*Flight Ticket*", and "*Airline*". Each task is associated with a set of relevant services. The returned services are sorted according to their similarity from high to low. The subject verified that the total of 18 tasks in the 5 processes listed in Table I can accurately reflect the specified goals.

TABLE I. CHARACTERISTICS OF THE GENERATED AD-HOC PROCESSES

Process ID	Ontologies/Goal	# of Concepts	# of Tasks
1	Travel	44	5
2	Watching Movie	28	4
3	Online Shopping	29	4
4	Credit Card Application	24	3
5	Stock Analysis	27	2

B. Evaluation of the Tag-based Service Description

We compare the performance of the proposed tag-based service in discovering Web services with a baseline approach which requires manually search for relevant services using keywords. We recruit a novice developer, who knows the general concepts of SOA and has limited experience in developing SOA systems. He manually searches for services to match with the tasks generated from the five ad-hoc processes. To compare our approaches with the baseline approach using the same set of tasks, we provide the 18 discovered tasks, which are generated from the prior study by the end-user subject, to the developer, who manually specifies keywords from their knowledge of the tasks as search criteria to query the service repository. The services are described using WSDL, without the tag-based service description as proposed in this paper.

We measure the effectiveness of both approaches in service discovery using *recall (r)*, *precision (p)*, *top-k precision* (p_k) and *r-precision* (p_r), as defined as follows.

$$p = \frac{|RetRel|}{|RetS|},\ r = \frac{|RetRel|}{|RelS|},$$
$$p_k = \frac{|RetRel_k|}{|k|},\quad p_r = p_{|RelS|} = \frac{|RetRel_{|RelS|}|}{|RelS|}$$

RelS is the set of relevant services; RetS is the set of returned services from a query; RetRel is the set of returned services that are relevant; and $RetRel_k$ is the set of relevant services in the top-k returned services. $RetRel_{|RelS|}$ is the number of relevant services at the top |RelS| number of returned services for a query.

Precision (p) is the ratio of the number of returned relevant services to the total number of returned services from the service repository. *Recall (r)* is the ratio of the number of returned relevant services to the total number of relevant services existed in the service repository. However, the number of the returned services can be too large for a user to review. Instead, a user would only go through the first *k* returned services. Therefore, we use the *top-k precision* and *r-precision* for our evaluation. The *top-k precision* evaluates the precision for the *top-k* returned services. For example, consider the case of getting 9 relevant services when 50 services are returned as a result of a query. Those 9 relevant services are listed in the top 10 returns, and there are 20 relevant services in total in the service repository. The *top-10 precision* is 9/10=90% whereas the regular precision would be 9/50=18%. The *R-precision* calculates the precision based on the number of relevant services at the top *r* returned services, and r is the total number of relevant services in the service repository. In the prior example, the *r-precision* evaluates the precision of the top 20 returned services since there are 20 relevant services in the entire repository. The *r-precision* in this example would be 9/20=45%.

TABLE II. RECALL AND R-PRECISION COMPARISON

	Recall	***R-precision***
our approach	0.98	0.63
Baseline	0.66	0.27

To calculate the *recall* and *r-precision*, one graduate student spent around 2 weeks in manually analyzing the 504 Web services registered in the service repository and to identify the relevant services for each task. Table II lists the average recall of both approaches. In the searches for 18 tasks, our approaches can find all the relevant services with a recall of 98%. In the baseline approach, a few relevant services are not returned using the provided keywords since the developer does not use the same words as the WSDL description to search for Web services. In summary, our approach has a higher recall.

Table II also lists the average r-precision of each task for both approaches. In our approach, an ontology provides more detailed information for describing a service. We calculate the averages of the *top-k* precisions (ranging from *top-1* precision to *top-6* precision) for all the tasks. Figure 8 shows the results for average *top-k* precision for all 18 tasks,

when *k* ranges from 1 to 6. As shown in Figure 8, our approach outperforms the baseline approach. The ontology definition used in our approach captures the expert knowledge and provides more relevant search keywords for each task; and therefore increases the success of the service discovery.

We use the precision vs. recall graph to display the performance of both approaches as shown in figure 9. The ideal approach should achieve high precision and high recall. A good performance is indicated by the trend line of an approach appearing in the upper right portion of the graph shown in Figure 9. As shown in Figure 9, the precision rate of our approach decreases slower than the baseline approach as the recall increases. As a result, our approach demonstrates higher precision and recall.

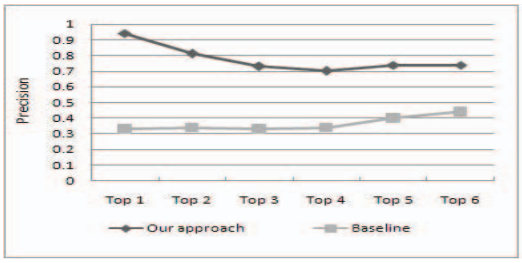

Figure 8. Top-k Precision

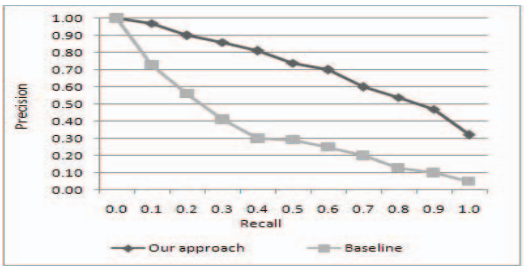

Figure 9. Recall vs. Precision Curves

C. *Discussion*

We observe that the baseline approach is highly dependent on the keywords provided by the novice developer and his domain knowledge. When a novice developer is not familiar with the application domain, the search with the provided keywords often returns no services although several relevant services exist in the service repository. Our approach uses the expert knowledge captured in an ontology and the tag-based services description. The service retrieval in our approach is independent from a user's familiarity with the domain and their knowledge of Web services. Therefore, our approach achieves high precision and recall in the service discovery. We received positive feedback from the end-user subject who used the prototype to compose the ad-hoc processes. The end-user subject can get relevant services automatically without having to search for the services over the Web. The services are organized in an abstract ad-hoc process which makes it easy for him to navigate through the services. Moreover, the end-user subject found that the tag-based service information very intuitive for understanding the functionality and usage of the services. The novice developer found it is challenging to understand the WSDL description for each service and select appropriate services without much prior experience.

Overall, our approach hides the complexity of the SOA standards and technologies from the end-users and enables end-users to participate in the service composition and adopt them into their daily on-line experience. However, we also note that the number of tasks generated is dependent on the number of relevant services in the repository. The generated process largely depends on the quality of the ontologies for the goal. If an ontology does not define the main concepts of the goal or does not represent concepts in a good structure, the generated process may include some tasks which the end user does not need or it might miss some important tasks which are necessary for achieving the goal.

VI. RELATED WORK

Enhancing service description with semantics. Current standards for Web services (e.g., WSDL) provide only the syntactic level description without semantic meaning. Semantic Web is proposed to enrich the service description with semantics. A few approaches, such as [21][25] enhance the Web service standards with semantics. Other approaches build a separate semantics description for web services. For example, the Ontology Web Language (OWL)-S [19] is built on top of WSDL to describe Web services using ontologies. DAML-S [2] uses DAML+OIL (the predecessor of OWL) based ontology to specify semantic service descriptions. Formalized semantic models are required for the semantic Web service descriptions. However, semantic Web services technology appears immature for the adoption into practices [12]. In a recent survey [13], the semantic Web service descriptions are limited compared to the service description for Web services. In our work, we enhance the service description using tags which are built on top of existing WSDL and ontologies. We do not require formal semantic models.

Automatic service composition. Well-defined business processes and Artificial Intelligence (AI) planner techniques are used to automatically compose Web services [17] [24][29]. Such techniques require formally describe tasks and the pre and post conditions for each task for achieving a user's goal. Therefore these techniques have limited support for the dynamic discovery of tasks. Our work can reflect the changing needs of an end-user without the pre-defined processes or tasks. Similar to our approach, ontologies are used for service discovery and service composition [3][4][11]. In [3], the services are classified using ontology to guide users to search for services. The approach in [4] uses ontology to semi-automatically compose services by matching the interface of individual services. Different from the aforementioned approaches, our approach identifies the

control relations among identified tasks and generates ad-hoc processes from ontologies.

Services Mashup. Service Mashup supports a lightweight way for service composition in a Web browser without the formal definition of business processes. Liu et al. [18] uses a Mashup model to assist developers to compose services. Carlson et al. [7] reuse service discovery approaches to find functionally equivalent non Web service based components, such as portlets, Web applications and widgets. The service markup is easier for non-expert user to learn and to manually compose services. Our work enhances service Mashup by providing guidance to end-users as they create their Mashups through the automatic composition of services.

VII. CONCLUSION

In this paper, we provide an approach that hides the complexity of SOA standards and tools from end-users and automatically composes services to help an end-user fulfill their daily activities. We propose a tag-based service description to allow users to understand the functionality of a service and add their own descriptive tags. Using our approach, an end-user only needs to specify the goal of their activities using keywords. Our approach automatically composes services that help an end-user achieve their desired goals without requiring the user to specify the detailed tasks. Our case study demonstrates the effectiveness of our approach for an end-user to compose services. Moreover, our approach can achieve higher precision and recall in the service discovery than the baseline approach.

Over time, the number of tags associated with a service would grow considerably, since any user can freely add new tags to the service description. Extraneous tags would negatively affect the effectiveness of the service discovery and end-users' understanding of the services. To reduce redundant tags, we plan to investigate techniques for checking semantically equivalent tags before adding them to the repository. We also plan to identify the conflicting tags for describing the same service. In the future, we plan to integrate Swoogle into our prototype to automatically search for ontologies. We also plan to conduct a larger user study to better evaluate the benefit of our approach.

REFERENCES

[1] D. Aguilar-Lopez, I. Lopez-Arevalo, V. Sosa-Sosa, "Toward the Semantic Search by Using Ontologies," Intl. Conference on Electical Engineering, Computing Science and Automatic Control (2008)

[2] A. Ankolekar, et al., "DAML-S: Web Service Description for the Semantic Web," Intl. Semantic Web Conference (2002)

[3] K. Arabshian, C. Dickmann and H. Schulzrinne, "Ontology-Based Service Discovery Front-End Interface for GloServ," LNCS In The Semantic Web: Research and Applications (2009)

[4] I. B. Arpinar, B. Aleman-Meza, R. Zhang, A. Maduko, "Ontology-Driven Web Services Composition Platform," IEEE Intl. Conference on E-Commerce Technology (2004)

[5] D. Beckett, B. McBride, RDF/XML Syntax Specification (Revised), W3C Recommendation (2004)

[6] S. Brin, L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," Intl. Conference on World Wide Web, p.107-117 (1998)

[7] M. P. Carlson, A. H. H. Ngu, R. M. Podorozhny, L. Zeng, "Automatic Mash Up of Composite Applications," Intl. Conference on Service Oriented Computing (2008)

[8] R. Chinnici, J. Moreau, A. Ryman, S. Weerawarana, "Web Service Description Language Version 2.0," W3C Recommendation (2007)

[9] Facebook, http://www.facebook.com/, last accessed on June 18, 2009.

[10] W. Fang, L. Zhang, Y. Wang, S. Dong, "Toward a Semantic Search Engine Based on Ontologies," Intl. Conference on Machine Learning and Cybernetics (2005)

[11] M. Flügge, D. Tourtchaninova, "Ontology-Derived Activity Components for Composing Travel Web Services," Intl. Workshop on Semantic Web Technologies in Electronic Business (2004)

[12] M. Klusch, "Semantic Web Service Description", book CASCOM: Intelligent Service Coordination in the Semantic Web, Birkhäuser Basel publishing, page 31-37 (2008)

[13] M. Klusch, Z. Xing, "Semantic Web Service In the Web: A Preliminary Reality Check", First Intl. Joint Workshop SMR^2 on Service Matchmaking and Resource Retrieval in the Semantic Web, Busan, South Korea (2007)

[14] IBM Mashup Center, http://www-01.ibm.com/software/info/mashup-center/, last accessed on June 18, 2009.

[15] IBM WebSphere Integration Developer, http://www-01.ibm.com/software/integration/wid, last access on June 18, 2009

[16] IBM WebSphere Service Registry and Repository, http://www-01.ibm.com/software/integration/wsrr/, last access on June 2, 2009

[17] U. Küster, M. Stern, B. König-Ries, "A Classification of Issues and Approaches in Service Composition," International Workshop on Engineering Service Compositions (2005)

[18] X. Liu, Y. Hui, W. Sun, H. Liang, "Towards Service Composition Based on Mashup," IEEE Congress on Services (2007)

[19] D. Martin, et al., "OWL-S: Semantic Markup for Web Services," Technical Report, Member Submission, W3C (2004)

[20] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, "Service-Oriented Computing Research Roadmap", European Union Information Society Technologies, Directorate D – Software Technologies (2006)

[21] P. Rajasekaran, J. Miller, K. Verma, A. Sheth, "Enhancing Web Services Description and Discovery to Facilitate Composition," Intl. Workshop on Semantic Web Services and Web Process Composition (2004)

[22] J. Rao, X. Su, "A Survey of Automated Web Service Composition Methods," Intl. Workshop on Semantic Web Services and Web Process Composition (2004)

[23] Seekda, http://seekda.com/, last accessed on June 16, 2009

[24] M. Sheshagiri, M. desJardins, T. Finin, "A Planner for Composing Services Described in DAML-S," AAMAS Workshop on Web Services and Agent-Based Engineering (2003)

[25] K. Sivashanmugam, J. A. Miller, A. Sheth, K. Verma, "Framework for Semantic Web Process Composition," Intl. Journal of Electronic Commerce, Winter 04/05 issue, 9 (2), pp. 71-106 (2004)

[26] M. K. Smith, C. Welty, McGuinness, D. L.: OWL Web Ontology Language Guide, W3C Recommendation (2004)

[27] Swoogle, http://swoogle.umbc.edu/, last accessed on August 12, 2009

[28] Web Services Business Process Execution Language, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, last accessed on June 2, 2009

[29] D. Wu, B. Parsia, E. Sirin, J. Hendler, D. Nau, "Automating DAML-S Web Services Composition Using SHOP2," Intl. Semantic Web Conference (2003)