

Ontology-driven service composition for end-users

Hua Xiao · Ying Zou · Ran Tang · Joanna Ng ·
Leho Nigul

Received: 19 May 2010 / Revised: 27 February 2011 / Accepted: 1 March 2011 / Published online: 24 March 2011
© Springer-Verlag London Limited 2011

Abstract Current service composition techniques and tools are mainly designed for use by Service-Oriented Architecture (SOA) professionals to solve business problems. Little attention has been paid to allowing end-users without sufficient service composition skills to compose services and integrate SOA solutions into their online experience to fulfill their daily activities. To shelter end-users from the complexity of service composition, we propose an approach which can compose services on the fly to meet the situational needs of end-users. We present a tag-based service description schema which allows non-IT professional users to easily understand the description of services and add their own descriptions using descriptive tags. Instead of requiring end-users to specify detailed steps for composition, the end-users only need to describe their goals using a few keywords. Our approach expands the meaning of a user's goal using ontologies then derives a group of keywords to discover services in order to fulfill the goal. A prototype is developed as a proof of concept to show that our approach enables end-users to discover and compose services easily. We conduct a case study to evalu-

ate the effectiveness of our approach that eases end-users to compose services without the knowledge of SOA technologies. The results of our case study show that our approach can effectively generate ad-hoc processes and discover services with relatively high precision and recall.

Keywords Ontology · Service-oriented architecture · Service composition · Service discovery · Web services

1 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 online services to perform repeated activities, such as online 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. Currently, an end-user needs to manually browse different Web services, which provides specialized services to perform each of the tasks in order to plan a trip. For example, expedia.com is a specialized website for booking hotels and checking flight tickets. However, one website seldom provided sufficient information to complete the entire trip planning. A user often visits other websites to get additional information to help him/her to make a decision. For example, a user would visit <http://www.theweathernetwork.com> to check the weather of destination and find the reviews of the hotels using <http://www.tripadvisor.com>. When planning the next trip,

H. Xiao (✉)
School of Computing, Queen's University, Kingston, ON, Canada
e-mail: huaxiao@cs.queensu.ca

Y. Zou · R. Tang
Department of Electrical and Computer Engineering,
Queen's University, Kingston, ON, Canada
e-mail: ying.zou@queensu.ca

R. Tang
e-mail: ran.tang@queensu.ca

J. Ng · L. Nigul
IBM Toronto Lab, Markham, ON, Canada
e-mail: Jwng@ca.ibm.com

L. Nigul
e-mail: lnigul@ca.ibm.com

the end-user needs to repeat the same activities. 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. It would be ideal if end-users can compose and customize their ad-hoc processes to fulfill the goals of their daily activities.

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-IT professional 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 three challenges:

1. *Complexity of service descriptions.* Service descriptions specify the capability of services, which usually includes input/output parameters, exceptions, functional and non-functional description. For example, Web Service Description Language (WSDL) [11] 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. OWL-S [31] uses ontologies to describe the semantics of Web services, especially the functionalities. However, existing service descriptions are too complex for non-IT professional end-users to understand. WSDL and OWL-S are primarily intended for SOA professionals instead of non-IT professional end-users to understand the interface and functionalities of a Web service.
2. *Complexity of service composition languages.* Service composition languages, such as Business Process Execution Language (BPEL) [25], are designed for SOA professionals to assemble services to form well-defined business processes. Service composition languages require very formal descriptions in different perspectives, such as variables, control flow, fault handling, and service binding. Although BPEL process modeling tools are provided to visualize service composition languages, such as IBM WebSphere Integration Developer (WID) [23], Oracle BPEL Process Manager [34], and ActiveBPEL [1], those tools are designed for SOA professionals instead of non-IT professional end-users. For example, the tools require users to understand different components of BPEL before the users can design a BPEL process. In addition, the ad-hoc processes needed by end-users for daily activities generally compose services in a loose way without strict execution order which cannot be described by existing service composition languages. For example,

when planning a trip, the end-user can buy the flight ticket first then book a hotel, and vice versa.

3. *Limited support for composing services on the fly.* The ad-hoc process needed by end-users generally requires the dynamic integration of various services (e.g., Web services and websites) on the fly. As aforementioned, a system integrator can specify BPEL processes to compose Web services using tools, such as IBM WID [23]. After the deployment of a BPEL process, the composition logic can be hardly customized to accommodate changes to consumer's requirements, as this involves a long lifecycle from design, development, and testing to deployment. In addition to manual composition by professionals, automatic service composition is a promising technique to dynamically compose services. However, most of the existing automatic service composition techniques require semantic Web service descriptions [29, 44, 48]. Different semantic service description languages, such as OWL-S [31] and Web Service Modeling Ontology (WSMO) [9], have emerged from the research community. However, most semantic service description languages still assume that the service consumers would share the same domain model (e.g., ontologies) with the services providers to a certain extent [48]. In practices, most of the services consumers and service developers are independent and have incompatible domain models.

To help non-IT professional end-users compose services for their daily activities, we propose an approach that hides the complexity of Web services standards and tools. Our approach can identify services which reflect the situational needs of users. More specifically, we address the aforementioned challenges in the following three 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 descriptive tags are easy to understand by end-users. Meanwhile, end-users can provide feedback based on their experience of using the services. Our service description approach creates a uniformed format to describe services with various service descriptions and make them accessible for end-users.
2. To describe the loosely coupled services in ad-hoc processes, we propose an ad-hoc process model. The ad-hoc process model records the tasks performed by end-users to fulfill their needs. Each task in the ad-hoc process is associated with a set of services which have similar functionality. To ease the end-users to navigate tasks in an ad-hoc process, we also suggest the possible control flows to reflect the navigational relations among tasks.
3. Instead of requiring end-users to specify the concrete tasks, the end-users only need to describe the goal that

they want to achieve. For example, the goal for planning a trip can be expressed using keywords, such as “Trip” or “Travel”. To derive the tasks that achieve the specified goal, we analyze the semantic meaning of the specified goal using ontologies. Ontologies capture the 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 “Tourist Attraction”. 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 provide an algorithm that analyzes the identified ontology to dynamically discover services and compose an ad-hoc process to achieve the specified goal.

This paper extends our earlier work [54] published in the International Conference on Service-Oriented Computing and Applications (SOCA) 2009. We enhance our earlier publication in SOCA 2009 in the following aspects:

1. Providing a generic ontology definition model to capture the information from different ontology languages;
2. Refining the ad-hoc process model to describe different control flows among tasks in order to guide an end-user to navigate through tasks;
3. Improving the algorithm for generating ad-hoc processes by inferring control flows among tasks from ontologies; and
4. Conducting more thorough case studies to evaluate different features of our approach.

The remainder of this paper is organized as follows. Section 2 introduces our generic ontology definition model.

Section 3 presents our approach to compose ad-hoc processes. Section 4 describes a proof of concept prototype for our approach. Section 5 discusses the case studies. Section 6 gives an overview of the related work. Section 7 concludes the paper and presents the future work.

2 An ontology definition model

An ontology expresses common entities (e.g., people, travel, and weather), and the relations among those entities. An ontology can be visualized as a graph that contains nodes representing entities and edges representing relations among the entities. Figure 1 illustrates an example ontology for defining the entity “Travel”. The entity “Travel” is related to four more specific entities: “Transportation”, “Accommodation”, “Tourist Attraction” and “Car Rental”.

Ontologies are defined using various ontology specification languages, such as Web Ontology Language (OWL) [46], Resource Definition Framework (RDF) [7], and DARPA Agent Markup Language (DAML) + Ontology Inference Layer (OIL) [13]. To capture the entities and general structures specified in various ontology specification languages, we summarize the commonality of different ontology specification languages using an ontology definition model. Figure 2 illustrates the main components of our ontology definition model and their relations.

- *Class*: is an abstraction description of a group of resources with similar characteristics. For example, “Hotel” is a *class* which describes the common characteristics of different types of hotels.
- *Individual*: is an instance of a *class*. *Individuals* are the objects and *classes* describe a set of *individuals*.

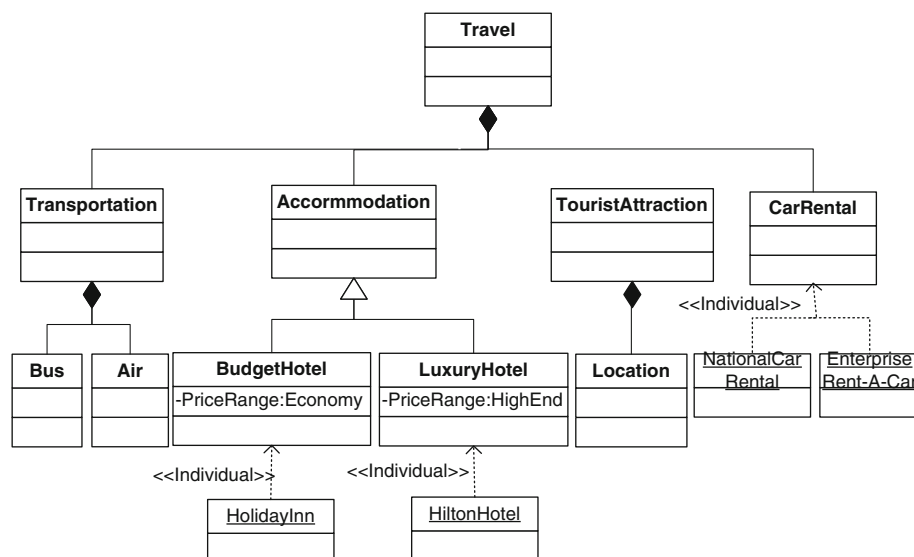


Fig. 1 An example ontology

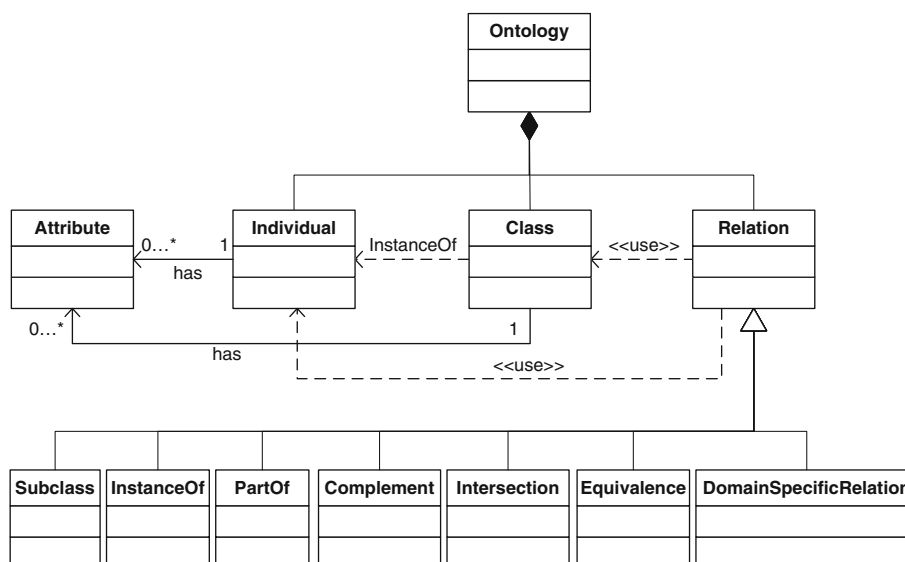


Fig. 2 Components of ontology definition model

- **Attribute:** is a property that a *class* and an *individual* can have. The attributes defined in a class are applied to the individuals.
- **Relation:** defines various ways that *classes* or *individuals* can be related to one another. For example, entity “BudgetHotel” is a subclass of “Accommodation”. The relations among classes are applied to the corresponding individuals. Figure 2 lists the common relations defined in various ontology specification languages. We summarize 7 types of relations among classes and individuals.
 - **Subclass:** extends an abstract *class* to convey more concrete knowledge. The *subclass* relation describes the parent and children relations among the connected *classes*. The *subclasses* can inherit the attributes in its parent class. As shown in Fig. 1, “Budget Hotel” and “Luxury Hotel” are the subclasses of “Accommodation”.
 - **InstanceOf:** describes that an *individual* belongs to a *class*. For example, individual “Hilton Hotel” is an instance of class “Luxury Hotel”.
 - **PartOf:** indicates that a class is a part of another class or an individual is a part of another individual. For example, class “Accommodation” is a part of class “Travel”. In some ontology specification languages, such as OWL, a class could be defined as a union of several classes, i.e., a class contains the distinct member classes. For example, “Weather” is the union of “Forecast”, “Wind”, “Temperature”, and “Precision”. To simplify the ontology definition model, we treat the union relation as a *PartOf* relation since the classes in a union relation is a part of the united class. For example class “Forecast”, “Wind”, “Temperature” and “Precision” is in a *PartOf* relation with class “Weather”.
 - **Complement:** selects all the *classes* from the domain that do not belong to a certain *class*. For instance, class “Budget Hotel” could be defined as a complement of class “Luxury Hotel”.
 - **Intersection:** define a *class* which has the common *individuals* of several classes. For example, “Luxury Hotel” is the intersection of “Hotel” and “Expensive Consumption”.
 - **Equivalence:** declares that two classes contain exactly the same individuals. For example, we can define class “Nation” is equivalent to class “Country”.
 - **DomainSpecificRelation:** specifies domain specific relations among classes. For example, airline company “Air Canada” is the sponsor of “2010 Winter Olympics”. Therefore, the relation between “Air Canada” and “2010 Winter Olympics” is “Sponsorship”. The ontology specification languages, such as OWL, use attributes to describe domain specific relations. To distinguish attributes from relations, we convert such a type of attribute description to *DomainSpecificRelation* in our ontology definition model.

3 An approach for composing ad-hoc processes

Figure 3 gives an overview of our approach for supporting end-users to compose ad-hoc processes. As shown in Fig. 3, the composition UI contains three parts: goal editor, process editor, and service selector. Goal editor is used to obtain the goal description from end-users. The process editor displays the generated ad-hoc process. Then end-users can use the process editor to customize the ad-hoc process, such as add a task and remove a task. Each task in the ad-hoc process is associated with one or more services. The service selector lists all

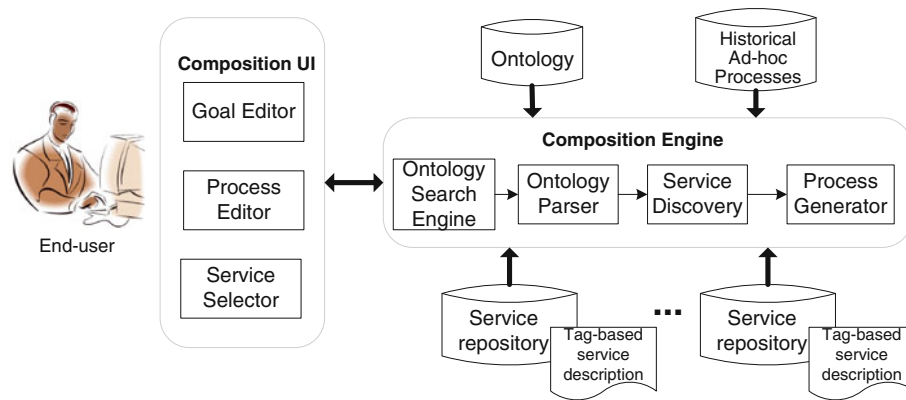


Fig. 3 An overview of our approach

the associated services when an end-user clicks a task. The end-user can select a service from the list to fulfill the task.

To compose an ad-hoc process, an end-user simply describes a desired goal using keywords in the composition User Interface (UI). The composition engine uses the goal description (i.e., keywords) to find a matching ontology from the ontology database. In an ontology, the semantics of a high-level goal is expanded into more concrete information, such as the attributes of the goal, the objects, and the actions (defined as classes in the ontology). We parse the matching ontology and extract keywords from the matching ontology to search for services in service repositories. The service repository allows service providers to advertise their services and provide interfaces for automatic service discovery. The services in service repositories are described using our tag-based service description schema which uses descriptive tags (i.e., keywords) to simplify the existing service descriptions, such as WSDL. The service description schema can help end-users and composition tools understand the properties of services. End-users can also edit the tags to refine the service description.

To facilitate the selection and execution of services, the process generator aggregates the discovered services into tasks. We use the relations defined in the ontology to identify the control flow among tasks. The identified tasks and control flows are represented as an ad-hoc process. The ad-hoc process can be stored in the ad-hoc process database and shared among multiple end-users.

In the following sub-sections, we discuss the details of the ad-hoc process model, the tag-based service description, the techniques to search for ontologies, services discovery, and ad-hoc process generation.

3.1 An ad-hoc process model

An ad-hoc process records the tasks that need to be performed for achieving a goal. Figure 4 shows the schema for representing an ad-hoc process. A task can be associated with

more than one service of the similar functions. For example, the task, “Car Rental”, can be fulfilled by different car rental companies. The end-users can choose their favorite services to fulfill the task. The services fulfilling a task are either directly discovered from a service repository or composed by other ad-hoc processes (i.e., sub-processes). In some cases, a few tasks have to be performed in a particular order. For example, two tasks, “Select a Product” and “Add to Shopping Cart”, must be performed before the task “Checkout”. To help user to perform tasks, we define two basic relations among tasks in our ad-hoc process model:

- *And* relation indicates that all the tasks have to be executed. Some *and* relations can be further specialized as *Sequence* and *Parallel* relations.
 - *Sequence* defines a set of tasks to be executed in a sequential. For example, the task “Select Flight Tickets” needs to be performed before processing the payment for the flight tickets.
 - *Parallel* describes independence of tasks, i.e., tasks in a parallel relation can be executed in any order or in the same time. For example, a user can “Book a Flight Ticket” and “Check the Weather Forecast” in parallel.
- *Or* relation means that users only need to execute one task from a given set of tasks. An *or* relation is further specialized to *Alternative* and *Choice* relations.
 - *Alternative* allows users to select one task among two tasks. For example, the user can select a favorite transportation vehicle from “Car” and “Train”.
 - *Choice* defines that users can select one task from more than two tasks.

3.2 Tag-based service description

To generate a meaningful ad-hoc process, it is critical to describe services in an efficient way that allows end-users

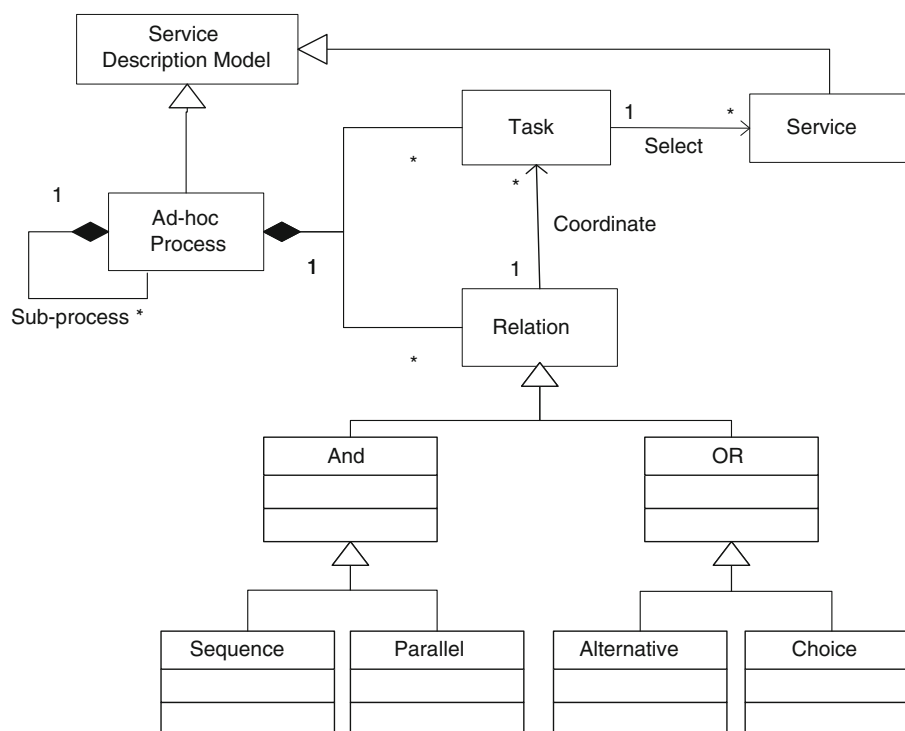


Fig. 4 Description of an ad-hoc process

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 [16] uses tags to describe images and Seekda [42] 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 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 which capture various properties of a service provided by both end-users and service providers to reflect the different perceptions of a service. To ensure tags reflect the functionality of the services, the tags are initially extracted from WSDL. End-users can add new tags.

3.2.1 Schema of the tag-based service description

Figure 5 illustrates the schema for our tag-based service description. In the schema, service providers describe the technical specification of services. End-users can provide their feedback on the quality of services and the delivered functionality. In general, we classify the tags into three categories:

General description specifies the basic characteristics of a service, such as version number, the last modified time, and

the URL address (i.e., link) for accessing the service. The general description is provided by a service provider.

Functional description is provided by both service providers and end-users to describe the functionality of a service. The functional description is composed by a set of operations. Each operation contains several detailed descriptions, such as input, output, constraints, user reviews, and provider's description on the functionality of an operation. 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. An end-user can add their own descriptions about each operation using a set of keywords. Many end-users may submit similar reviews. Similar to the indexing techniques used in existing search engines [8], we extract the meaningful keywords from the reviews and store them as tags.

Quality of services 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)

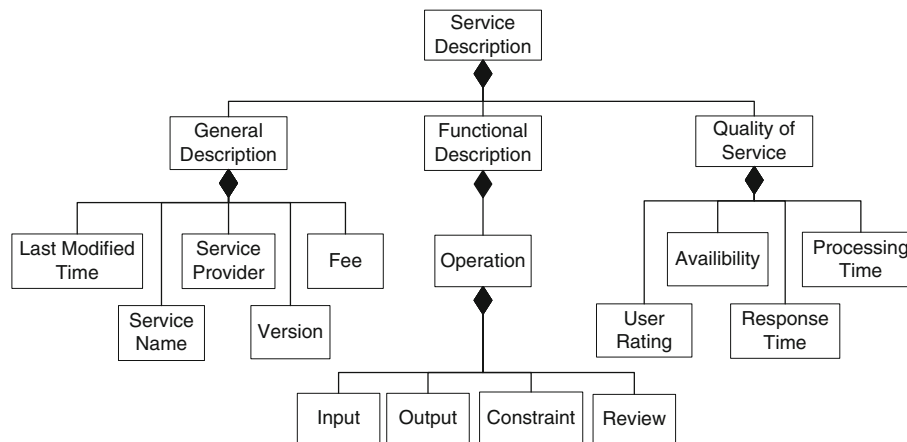


Fig. 5 Schema for tag-based service description

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 existing description documents of a service (e.g., WSDL) with the tag-based description.

3.2.2 Management of tags

The WSDL description for a service is provided by a service provider. We analyze the WSDL description of a service to extract the tags required from service providers. For example, we extract the service name, operation names, input parameters, output parameters, and service access location (i.e., the URL to access the service) from WSDL documents. Figure 6 shows an example WSDL description, which contains two operations, namely, “GetWeatherByZipCode” and “GetWeatherByPlaceName”. The name of the service is “WeatherForecast”. To extract meaningful tags from the user’s input and WSDL files, we filter out stop words, such as “the”,

“to”, and “by” from the input phrases. When a stop word is placed in the middle of a phrase, we use the stop words to separate the phrase into two tags. For example, the operation name “GetWeatherByZipCode” is converted to <Get Weather> and <Zip Code>.

Instead of treating derived words (e.g., traveling and traveled) as different tags, we detect the derived words from the user’s input and WSDL documents, and store only the corresponding root form in the description. For example, the derived words, such as “Travelling”, “Traveled”, and “Travels”, are stored using the root word “Travel” in the service description. Not all the tags in our schema can be extracted from WSDL documents since WSDL is designed to provide the programming interfaces and does not have the semantic descriptions on the functionalities and QoS. We allow end-users and service providers to add more tags by providing either phrases or sentences to describe their reviews of a service. Over time, tags associated with a service would grow considerably large since any users can freely add new tags to the service description. Extraneous tags would negatively affect the effectiveness of service discovery. To reduce the redundant tags, we use WordNet [51] to detect the synonyms of a new tag to be added in the existing tag-based description.

```

+ <wsdl:message name="GetWeatherByPlaceNameHttpPostOut"></wsdl:message>
- <wsdl:portType name="WeatherForecastSoap">
  + <wsdl:operation name="GetWeatherByZipCode"></wsdl:operation>
  + <wsdl:operation name="GetWeatherByPlaceName"></wsdl:operation>
  </wsdl:portType>
  :
- <wsdl:service name="WeatherForecast">
  + <documentation></documentation>
  + <wsdl:port name="WeatherForecastSoap" binding="tns:WeatherForecastSoap"></wsdl:port>

```

Fig. 6 An example of the service description in WSDL

Input: Goal description; task description (option)
Output: A matching ontology
Procedure searchOnto () {
 1. **Var** ontoSet = null;
 2. Goal description = goal description \cup { the synonyms of the keywords in goal description };
 3. **For each** keyword k_i in goal description {
 4. Search for ontologies matching with k_i ;
 5. If(find matching ontologies)
 6. { Add the matching ontologies to ontoSet; }
 7. } //end for
 8. If (ontoSet is empty) // Cannot find matching ontologies.
 9. { Return null; }
 10. If (the size of ontoSet == 1)
 11. { Return the ontology in ontoSet; }
 12. else{ //ontoSet contains more than one matching ontology;
 13. Sort the selected ontologies from high to low using the frequency of keywords in goal description and task description;
 14. Return the first ontology in the sorted ontology list;
 15. } //end else
 16. }

Fig. 7 Algorithm for searching for ontologies

We filter out semantically equivalent tags before adding them to the tag-based description document.

3.3 Search for ontology

Figure 7 shows our algorithm that uses keywords to search for ontologies. An end-user describes a process as a goal using a collection of keywords (i.e., $\text{keyword}(G) = \{k_1, k_2, \dots, k_n\}$, 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., $\text{keyword}(\text{plan trip}) = \{\text{travel}, \text{trip}\}$. An ontology is defined to capture the expert knowledge of a user-specified goal. Instead of predicting the possible users' goals and predefining the corresponding ontology, we can search for relevant ontologies on the Web. This allows an end-user to specify any goals for the ad-hoc processes. To improve the chances of discovering an ontology, we also enhance the user provided keywords with the set of synonyms of each keyword provided by a user's as the goal description. The keywords and the associated synonyms are collected as keyword set for a goal (i.e., $\text{keyword}(G)$). As an optional choice, we allow end-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 $\text{keyword}(T) = \{tk_1, tk_2, \dots, tk_m\}$ where tk_i refers to the keyword for describing a task.

An ontology of the specified goal is identified when the root entity of an ontology matches one of the keywords in $\text{keyword}(G)$ (i.e., the set of keywords for goal description). Root entity is the entity in the ontology graph which does not have parent node (e.g., node "Travel" in Fig. 1). For example

shown in Fig. 1, when the root class "Travel" is matched with a "Plan a Trip" goal description, the ontology is returned. If a goal description is matched with an entity defined within an ontology instead of the root entity, we retrieve the matched entity and its children nodes (e.g., shown in Fig. 1, the node "Accommodation" and all the related children nodes, such as "BudgetHotel" and "LuxuryHotel"). 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 of different meanings in both sets of the task description (i.e., $\text{keyword}(T)$) and the goal description (i.e., $\text{keyword}(G)$) appearing in each ontology. We select the ontology with the highest frequency of the provided keywords.

If the keywords in a goal description (i.e., $\text{keyword}(G)$) are phrases with more than one word, we search for matching ontologies using the entire phrase. However, we may not be able to locate any ontologies using the phrases in $\text{keyword}(G)$. In our experience, adjectives and adverbs are constraints for describing nouns. The essential information is expressed as nouns in the goal description. Therefore, we identify adjectives and adverbs by querying a dictionary and remove the adjectives and adverbs from the phrases. We use the rest of nouns as keywords to search for ontologies. The removed adjectives and adverbs are used to refine the searching results by selecting the ontology which contains the adjectives and adverbs if there are more than one matching ontologies.

To improve the chances for identifying ontologies, WordNet is used to identify the synonyms of the keywords in $\text{keyword}(G)$. We combine all the keywords in the goal description and the task description to search for ontologies based on the frequency of provided keywords. Then we ask end-users to inspect the returned ontologies and select one.

To help users understand the ontology description, tools such as protégé [40] are used by a user to visualize ontologies.

3.4 Searching for services

The identified ontology can provide more detailed description about a user's goal. Essentially, the classes and individuals defined in ontologies capture the characteristics of the functional requirements for desired services that help achieve a user's goal. We use classes and individuals (i.e., entities) as criteria to search for the matching services. However, simply using a single entity in the search criteria may prevent us from discovering services since a single keyword provides limited knowledge. Similar to the search engines, which use the expanded query to search for the relevant documents [2, 17], we group the name of the entity, its attributes, and the relevant entities which have direct relations (e.g., *Subclass*, *Partof*, *Equivalent*, and *InstanceOf*) with the entity as a set of keywords, i.e.,

$$\begin{aligned} \text{entity}(e_0) &= \{e_0\} \cup \{e_1, e_2, e_3, \dots, e_m\} \cup \text{attr}(e_0) \\ \text{attr}(e_0) &= \{a_{e01}, a_{e02}, \dots, a_{e0p}\} \end{aligned} \quad (1)$$

where e_0 is the name of the entity. e_i , where $i = 1, 2, 3, \dots, m$, refers to an entity which has direct relation with e_0 (i.e., e_i directly inherits from e_0 , is a part of e_0 , is equivalent with e_0 , or is an InstanceOf e_0), and a_{e0j} is the j th attribute for entity e_0 , where $j = 1, 2, \dots, p$.

In Eq. (1), entities $e_1, e_2, e_3, \dots, e_m$, which are in *Subclass*, *PartOf*, *equivalence*, or *InstanceOf* relations with entity e_0 , extend the meaning of entity e_0 . Therefore, we use these entities e_1, e_2, e_3, \dots , and e_m to enhance the meaning of entity e_0 . To avoid obscuring the meaning of entity e_0 with too many details, we only take the entities which have a direct relation with entity e_0 . For example, the class “Travel” is expanded with a set of its components described as *partOf* relation in our example ontology definition model, i.e., $\text{entity}(\text{travel}) = \{\text{travel}\} \cup \{\text{transportation}, \text{accommodation}, \text{tourist attraction}, \text{car rental}\}$. However, we do not use entities “Budget Hotel” and “Hilton Hotel” to extend the meaning of “Travel” since these two entities have no direct relations with “Travel”.

Each entity e_i has its own set of synonyms, i.e., $\text{synonym}(e_i) = \{s_{i1}, s_{i2}, \dots, s_{in}\}$. For example, the concept, “Travel”, has a set of synonyms, such as trip and journey (i.e., $\text{synonym}(\text{travel}) = \{\text{trip}, \text{journey}\}$). To retrieve the relevant Web services from a service repository, we combine the entity set and synonym sets into a keyword set (i.e., $\text{entity_keywords}(e_0) = \text{entity}(e_0) \cup (\bigcup_{i=1}^m \text{syn}(e_i))$) to search for the matching services in a service repository.

$$\text{SIM} = \frac{\text{\# of matched keywords}}{|\text{n}|} \quad (2)$$

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

As discussed in Sect. 3.2, each service, s_j , in a repository is described by a set of tags that specify the general information and the functionality. Therefore, we have a set of keywords to describe a service using tags, i.e., $\text{ws-keywords}(s_j) = \{t_1, t_2, \dots, t_z\}$ where s_j is a service and t_i is a tag of the service. To discover a service, we count the number of matched keywords between the entity description keywords (i.e., $\text{entity-keywords}(e_0)$) in the searching criteria and the service description tags (i.e., $\text{ws-keywords}(s_j)$) in the service repository. The similarity degree of the entities and services are defined in Eq. (2). 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 entities 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 desirable for the returned services. Figure 8 summarizes the algorithm that searches for services.

3.5 Generating ad-hoc processes

The set of classes, individuals, and attributes defined in ontology definition model could be used as searching criteria to search for 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 an algorithm to organize the returned services in a logical structure (i.e., ad-hoc process) in the following steps: (1) identify a list of tasks which are associated with one or more functionally similar services; (2) identify the relations among the tasks; and (3) refine the generated ad-hoc process by merging similar tasks which have the same set of associated services and relations.

3.5.1 Identifying tasks

To group the services with similar functionalities, we design an algorithm to identify tasks for the ad-hoc process. The details of the algorithm are described in Fig. 9. In this algorithm, we take an ontology which matches with the goal description as the input. More specifically, the ontology is

Input: ontology, entity e_0
Output: relevant service list
Procedure searchServices () {
 1. Using formula (1) to get the keyword set $entity(e_0)$ of e_0 ;
 2. $entity_keywords(e_0) = entity(e_0) \cup (\bigcup_{i=0}^m syn(e_i))$;
 3. Search service repository by matching $entity_keywords(e_0)$ with the tag-based service descriptions;
 4. Using formula (2) to calculate the similarity degree between the query and service descriptions;
 5. Sort the relevant services based on the similarity degree;
 6. If (two relevant services have the same similarity degree)
 7. {Sort these services based on the QoS description provided by the tag-based service description. }
 8. Return sorted relevant service list;
 9. }

Fig. 8 Algorithm for searching for services

Input: Ontology model for the goal
Output: A set of tasks associated with services
Initiate: var E = the entity which matches the goal description (i.e., keywords);
Procedure identifyTask (var E) {
 1. If E does not have attributes, direct subclasses, sub-components (described by part of relation), equivalent entities, or instances {
 2. Return;
 3. }
 4. Use equation (1) and (2) described in section 3.4 to search for services for E from service repository)*;
 5. **If**(the number of matching services > 0){
 6. Associate the matching service to E, and convert E as a task in the ad-hoc process;
 7. Output task E;
 8. }
 9. Set(Er) = entities which have a direct relation with E;
 10. **If**(the size of Set(Er) equals to 0) {
 11. return;
 12. }
 13. **For** each element Ei in Set(Er) {
 14. identifyTask(Ei);
 15. }
 16. }
 * Entity E has matching services if and only if there exist returned services whose similarity degree is greater than a predefined value.

Fig. 9 Algorithm of identifying task list

specified using our ontology definition model and represented as an ontology graph as shown in Fig. 1. In the ontology graph, the path length of two nodes is the shortest distance (i.e., the number of edges) along the path from one node to another. For example shown in Fig. 1, the path length of entities “Travel” and “Luxury Hotel” is 2.

In general, our algorithm uses a stepwise approach to discover and organize the tasks according to the level of abstraction. The high level entities in an ontology graph convey more abstract meanings that are suitable to discover the general purpose tasks. Such tasks allow end-users to receive the desired services (e.g., expedia.com) in one server without having to go through separate servers for different services. The low level entities in an ontology graph provide more specific meanings of the goal and therefore, indicate

the possibility to find concrete tasks which provide more specialized services. For example, expedia.com provides a general service for planning a trip by providing information on car rental, flight ticket purchasing, and hotel reservation. To check into a flight by Air Canada, an end-user has to visit more specialized services by going to Air Canada Web Site to print their boarding passes and check flight status. To satisfy an end-user with varying needs in different levels of specialization, we use the breadth-first search algorithm to scan the ontology graph. We identify general purpose tasks from the top of the graph and the specialized tasks from the low level of entities in the graph.

The algorithm starts from the entity that matches with the high level goal description to find general purpose services. Subsequently, we identify more concrete entities by visiting

the entities with the path length of 1 from the start point. Using the breadth-first search algorithm, we can identify a set of entities with different level of specialization by gradually increasing the path length from the start point. The algorithm converts a visited entity into a task if and only if the entity has at least one *attribute*, *subclass*, *sub-component* (in the *PartOf* relation), *equivalence* entity, or *instance*. The visited entities that cannot meet the condition are not converted as tasks since such entities contain too little information to find appropriate services.

For each identified task, our algorithm extracts the searching criteria (i.e., *entity-keywords* (e)) from the corresponding entity and searches for services. We derive the name of a task from the name of the corresponding entity. When the similarity degree between the keywords in the searching criteria and the tags in a service description is greater than a predefined threshold, the task is matched with the service. The task is ignored if no matching services can be found. The discovered services for a task are sorted based on the similarity degree. Finally, the algorithm outputs a list of identified tasks.

Figure 10 gives an example of generating task list using our algorithm. The algorithm starts from the class “Travel” which is the goal specified by the end-user. The searching criteria for services are formed from the name of class “Travel” as well as the names of entities which have the direct relations (i.e., *PartOf* relation in our example) with class “Travel” (i.e., “Transportation”, “Accommodation”, “TouristAttraction”, and “CarRental”). Each returned Web service is expected to provide a set of comprehensive services (e.g., similar to Expedia.com [15]) for end-users to plan the travel. However, the returned Web services for “Travel” may only provide limited services, such as booking flight and train tickets and renting car, but without bus information. In this case, the end-users may need more specialized services for transportation, accommodation, and tourist attractions. To offer end-users with more options, our algorithm keeps on decomposing the abstract goal into a set of more concrete tasks as the algorithm traverses deeper in the path. In the second iteration of our algorithm, our algorithm identifies tasks, such as “Transportation”, “Accommodation”, “TouristAttraction”, and “CarRental”. In the third iteration, our algorithm further refines Accommodation task with more specific tasks, such as “BudgetHotel” and “LuxuryHotel”. However, the entities, such as class “Bus” and “Air”, do not have sufficient information (e.g., *attributes*, *subclass*, *sub-component*, *equivalence* entities, or *instances*) to discover new services. Therefore, their parent node “Transportation” is not further decomposed to more specialized services.

3.5.2 Identifying the relations among tasks

Once we identify a set of initial tasks in an ad-hoc process, we derive relations (i.e., sequence, parallel, and choice rela-

tions) among tasks by analyzing the relations among the corresponding entities used for identifying tasks. Table 1 shows the mapping from entity relations in an ontology to task relations in an ad-hoc process. In Table 1, E_i is an entity defined in an ontology, and T_i is the corresponding task identified by entity E_i and its related entities in an ontology graph. As shown in Table 1, we convert the relations from the ontology to the ad-hoc process using the following mapping rules:

1. If entities E_1, E_2, \dots, E_n are the *subclass* of entity E , it indicates that E_1, E_2, \dots, E_n contain more detailed information regarding E . Therefore, we can replace E with E_1, E_2, \dots, E_n . The tasks T_1, T_2, \dots, T_n corresponding to entities E_1, \dots, E_n have a *choice* relation with task T corresponding to E . In addition, tasks T_1, T_2, \dots, T_n are in *choice* relations since they have the similar functionalities conveyed in entity E .
When entities E_1, E_2, \dots, E_n are in *PartOf* relation with entity E , it indicates that entity E is composed by entities E_1, E_2, \dots, E_n . Theoretically, *PartOf* relation cannot guarantee that entity E does not contain the parts other than entities E_1, E_2, \dots, E_n . For example, we define that “Bus” and “Air” are part of “Transportation” in an ontology, but “Transportation” may also contain other components, such as train and ferry. In our approach, we ignore those parts of E since those parts do not define in the ontology. Similar to the subclass relation, we substitute entity E with entities E_1, E_2, \dots, E_n . Therefore, we define tasks T_1, T_2, \dots, T_n are in a *choice* relation with task T . Moreover, tasks T_1, T_2, \dots, T_n need to be executed together to fulfill the task T , i.e., tasks T_1, T_2, \dots, T_n have a *parallel* or *sequence* relation. We will discuss how to distinguish *parallel* and *sequence* relation later in this section.
2. *InstanceOf* relation is similar to subclass relation since any instance contains the functionalities defined in the corresponding class.
3. *Complement* relation means that two entities cannot co-exist. Therefore, we convert it into *alternative* relation.
4. *Equivalence* indicates that two tasks have the similar functionality, so we convert it into *alternative* relation.
5. *Domain specific relations* vary from domains. The mappings between the domain specific relations and the task relations need to be manually specified.

Figure 11 shows an example relations identified from the task lists described in Fig. 10. The end-user can either choose a general purpose service “Travel” to fulfill the goal of planning the trip or decide to accomplish the goal using more specialized services (i.e., services for transportation, accommodation, tourist attraction, and car rental). For the accommodation, the end-user can choose a general service related

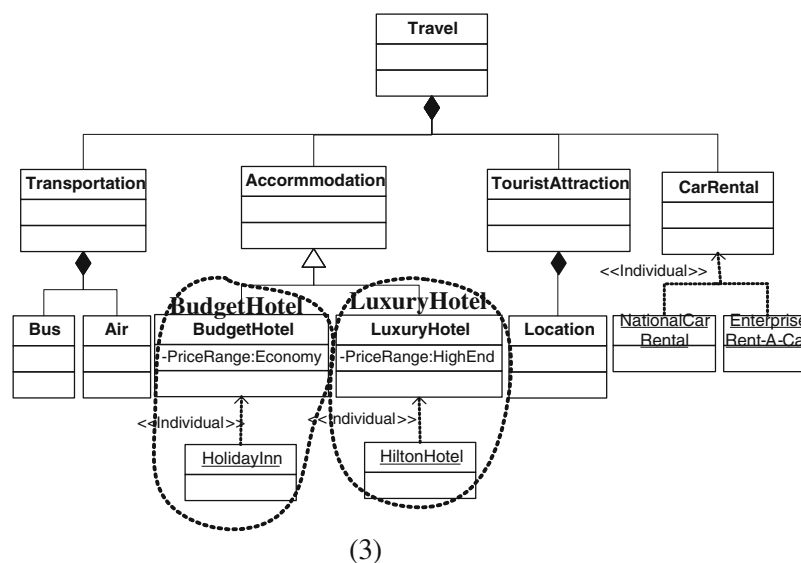
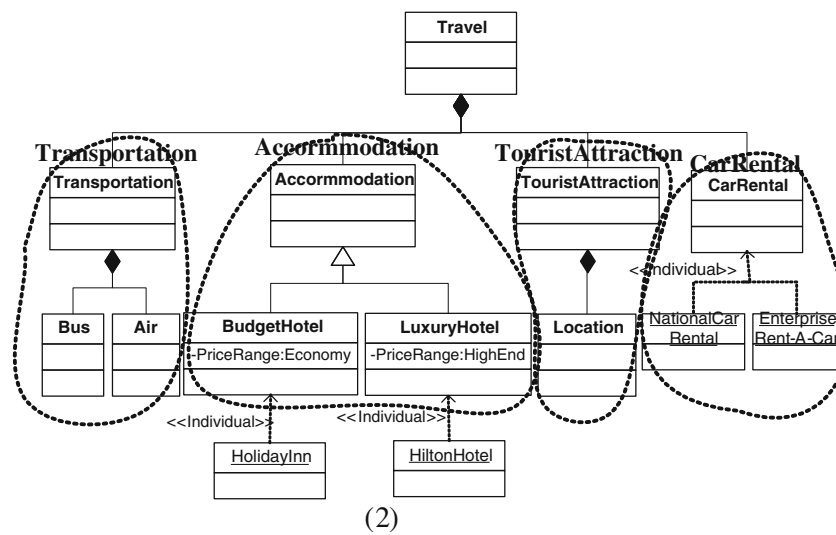
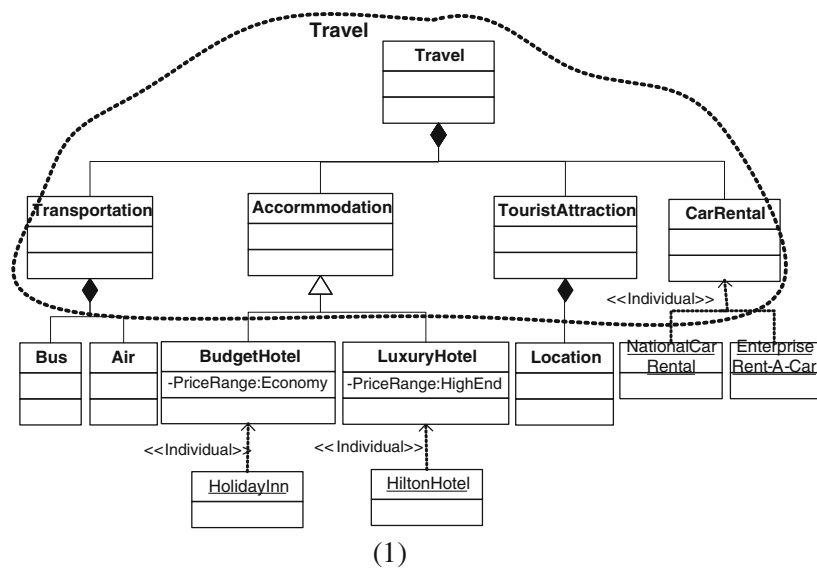
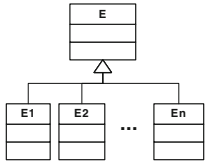
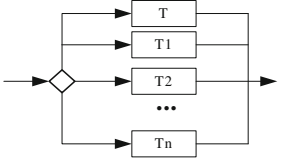
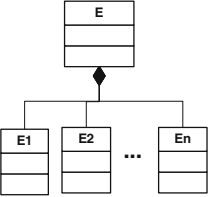
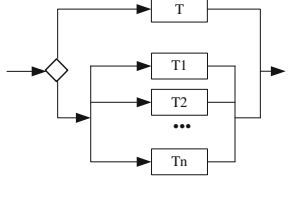
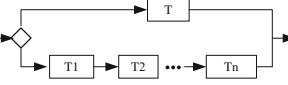
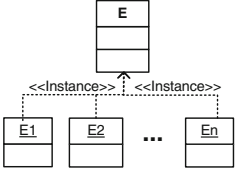
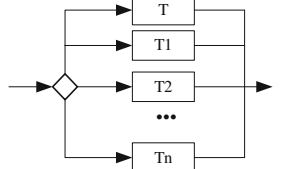
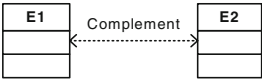
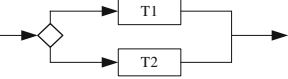
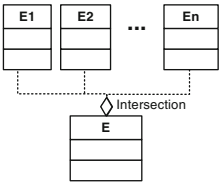
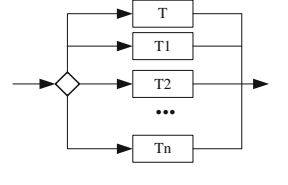
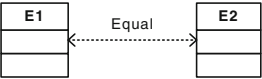
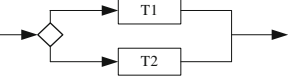


Fig. 10 An example of generating task list

Table 1 Convert relations from ontology to ad-hoc process

Relation in ontology definition model	Ontology graph	Control Flows in ad-hoc process
Subclass		
partOf		 OR 
InstanceOf		
Complement		
Intersection		
Equivalence		

to accommodation, or select a service offering specific types of hotels (i.e., budget hotel and luxury hotel).

Sequence indicates that the tasks are executed in sequence. *Parallel* means that the tasks can be executed in any order. To distinguish between *sequence* and *parallel* relations, we compare the interfaces of services associated to tasks. Assume that we have tasks T_1, T_2, \dots, T_n which could be either in *sequence* or *parallel* relations, and we need to distinguish the *sequence* and *parallel* relations. For each task, we collect all the input and output parameters from all of its associated services. We use *Set(input)* to represent all the input parameters

from the associated services of a task, and use *Set(output)* to represent all the output parameters from the associated services of a task. Then we match the interfaces of the corresponding services. Enlighten by Paolucci's work [37], we define three levels of matching as follows. Suppose that p_1 and p_2 are the input or output parameters of tasks T_1 and T_2 .

- **Exact:** if $Set(p_1) = Set(p_2)$
- **plug in:** if $Set(p_1)$ is a subset of $Set(p_2)$
- **Subsume:** if $Set(p_1)$ subsumes $Set(p_2)$, in other word, $Set(p_1)$ contains all the parameters in $Set(p_2)$.

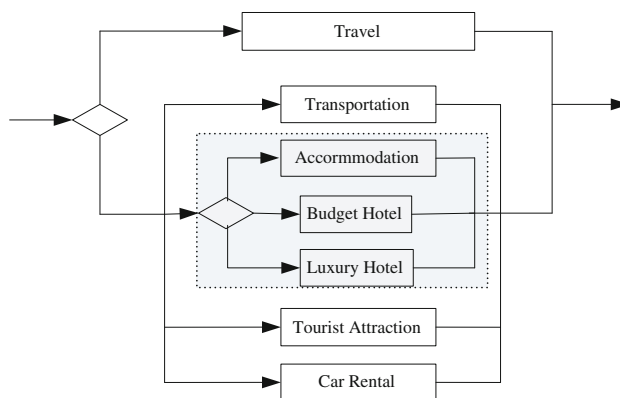


Fig. 11 An example of a generated ad-hoc process

If two sets of parameters satisfy one of the three conditions, then these two sets of parameters match. Two tasks T_1 and T_2 are in a sequence relation, if and only if the interfaces of the corresponding services can be matched (i.e., $Set(output_{T_1})$ matches with $Set(input_{T_2})$). We treat the tasks which are not identified as the *sequence* relation as *parallel* relation.

3.5.3 Merging tasks

In our algorithm, the generated ad-hoc process may contain tasks with very similar functionalities. For example, two tasks which are generated from two *equivalence* classes, respectively, may have very similar functionalities. Generally, similar tasks are converted into the *choice* control flow to allow end-users to select. However, if two similar tasks also associated with the same set of services, it becomes meaningless to offer the option to end-users. We use the following two rules to identify and merge similar tasks. Suppose that we have two tasks T_1 and T_2 , $WS-Set(T_1)$ is the set of associated services for task T_1 and $WS-Set(T_2)$ is the set of associated services for task T_2 .

Rule 1 if tasks T_1, T_2 are in a *choice* relation, and the overlap of $WS-Set(T_1)$ and $WS-Set(T_2)$ is greater than 90%, we choose the task that has more abstract meaning as defined in ontology to represent the merged task and associate services from both tasks to the merged task.

For example, if tasks “Accommodation” and “Hotel” are in a choice relation and associated with over 90% common services, we use task “Accommodation” to replace the two tasks and associate the relevant services of these two tasks to the task “Accommodation”.

Rule 2 If $WS-Set(T_1)$ is a subset of $WS-Set(T_2)$ (i.e., $WS-Set(T_1) \in WS-Set(T_2)$), then task T_1 is covered by task T_2 . We remove task T_1 .

4 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) [24] to register and manage Web services. To display the interfaces of selected services and invoke services, we use the IBM Mashup Center [21] as a platform to integrate various Web services. The Mashup platform allows end-users to customize the generated ad-hoc processes. Specifically, a Mashup page is a lightweight Web application that integrates multiple data or functions into one page to create a new service. A Mashup page provides an easy way for non-IT professional end-users to learn and manually compose services. For example, to facilitate shopping, an end-user can easily integrate online flyers, retailer’s websites, maps, and routes of public transportation into a single Mashup page by dropping and connecting different data resources together.

Figure 12 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 [47], 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 Fig. 12, once a user selects the “Car Rental” task in the *Process Editor*, the associated services are 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

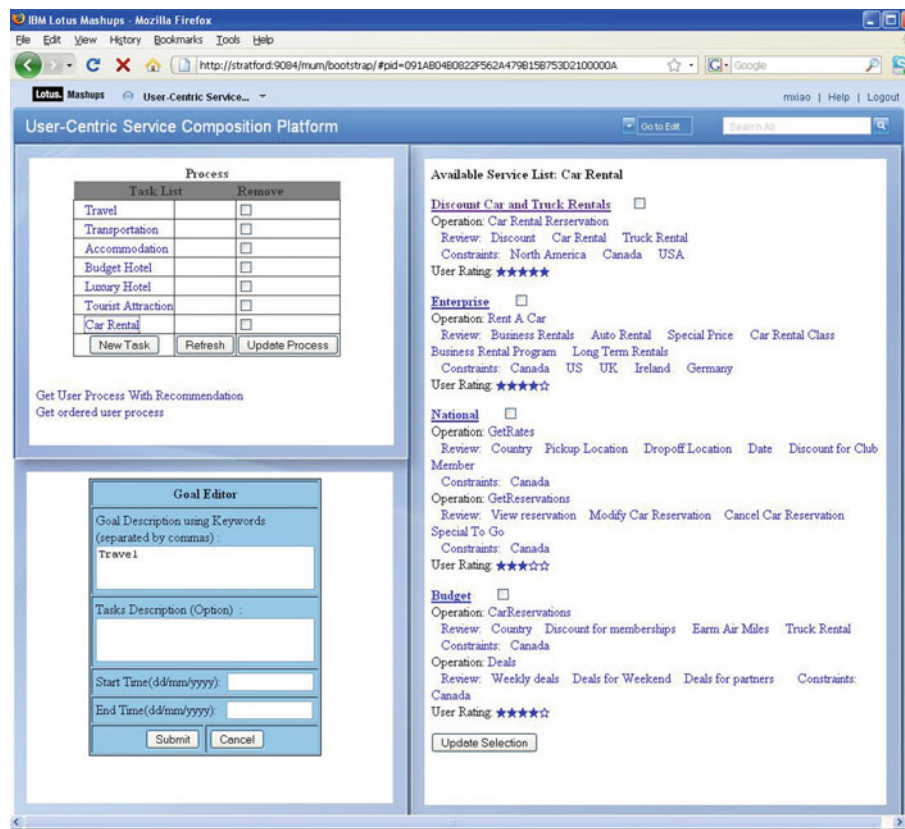


Fig. 12 Annotated screenshot for our prototype

preferences. When a user specifies the same goal, our prototype provides the previously refined ad-hoc process. To guide end-users to navigate through the task list to fulfill the goal, we identify the control flow among tasks based on the relations defined in the ontology. Figure 13 shows a screenshot of the suggested execution order of tasks in the ad-hoc process. Since the order ad-hoc process is only a suggested execution order, end-users still have the option to execute the tasks in other order. In [53], we present the details on recording these execution orders and refining the ordered ad-hoc process.

5 Case studies

We conduct a case study to evaluate the effectiveness of our approach that can support end-users without the knowledge of SOA technologies to compose services. Our approach can generate ad-hoc processes for end-users and discover the relevant services to fulfill the tasks in the ad-hoc processes. The objectives of our case studies are to examine (1) if the ad-hoc process generated by our approach can reflect an end-user's goals; (2) if the tag-based description and the service searching criteria extracted from ontologies can help locate the relevant services with high precision and recall; and (3)

if end-users is satisfied with the experience of the service composition.

5.1 Experiment setup

5.1.1 Collecting ontologies

In our case studies, we create an ontology database and use two approaches to obtain ontologies. One approach is to retrieve ontologies over the Web. We collect the online ontologies from Freebase [19], DBpedia [14], and Swoogle [47]. Freebase and DBpedia are two ontology databases which extract structured information from Wikipedia [50]. Freebase allows online users to edit and update the ontologies. Swoogle is an online ontology search engine. Freebase and DBpedia are focused on defining an entity without describing the steps for accomplishing a given goal. However, for some goals, we are not able to find the matching ontologies from the Internet.

The other approach overcomes the problems with the availability online ontologies. We use an ontology learning tool, named Text2Onto [12] to automatically learn ontologies from documents. More specifically, Text2Onto implements a variety of algorithms to learn ontologies from textual resources, such as Web pages, text files, and XML documents.

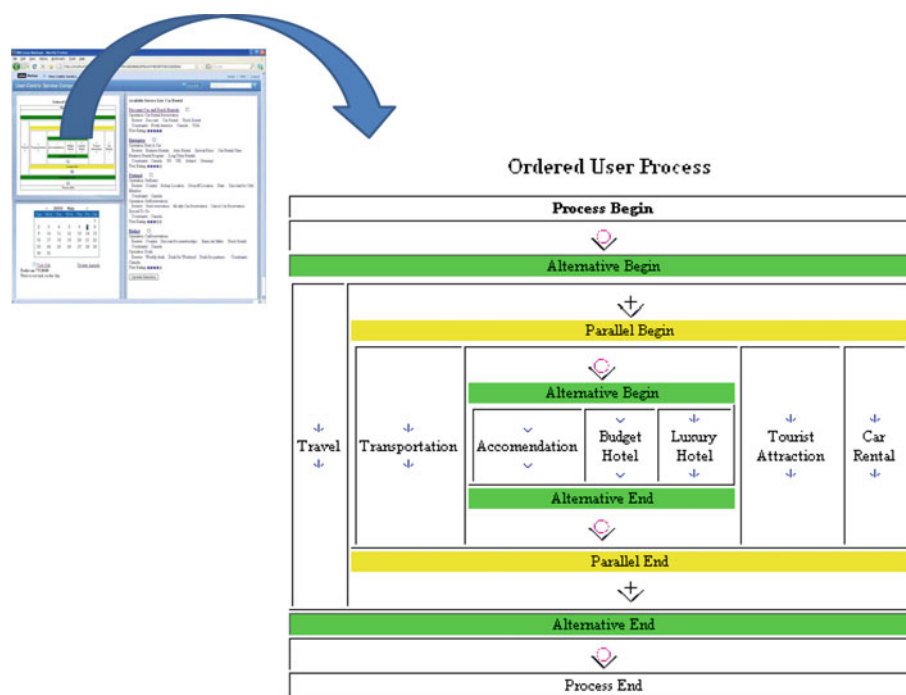


Fig. 13 Ordered ad-hoc process in the Mashup page

Table 2 Characteristics of generated ad-hoc processes

Domain	# of ad-hoc processes	# of tasks
Shopping	6	79
Travel	3	23
Banking	3	17
Entertainment	3	15
Others	5	59

We use Google [20] to search for the relevant Web pages, and run Text2Onto to learn the ontologies from those Web pages. However, the Text2Onto cannot delete the advertisements and irrelevant information on the Web page. We manually filter out the irrelevant information from the Web pages before importing them into Text2Onto to extract the ontologies.

In the case studies, one of the co-authors specifies 20 goals and each goal requires a certain process to achieve. Then we use the 20 goals to find 20 different ontologies from our ontology database. The 20 goals describe daily activities in different domains, such as shopping, travel, banking, and entertainment. Table 2 summarizes the distribution of the ad-hoc processes and the number of tasks identified in each domain after we use our approach to generate the ad-hoc processes.

5.1.2 Creating a web service repository

To collect Web services and create a service repository, we manually searched for Web services from the Internet using

Seekda [43] and Google. Seekda is an online Web services search engine. We registered 1,000 Web services into our service repository. Each service in the repository is described using our proposed tag-based service description. The services of different domains are selected to ensure that we can find services that match with the users' goals specified in the case study.

5.1.3 Subjects in the case studies

To evaluate the generated ad-hoc processes, we recruited 8 end-users to participate in our experiment. Nielsen [32] suggests that the best user study for gathering qualitative measures should involve three to five users. We carefully select the 8 subjects to make sure that they do not have any background on SOA since they are intend to represent the group of non-professional end-users. Four of the 8 subjects do not have background on SOA but are very familiar with Internet. The remainder 4 subjects do not hear SOA at all and they have the basic knowledge about Internet.

To evaluate the performance of our proposed tag-based service description in the service discovery, we use a baseline approach which requires a developer manually search for relevant services using keywords as a comparison. We recruit a novice developer, who knows the general concepts of SOA and has limited experience in developing SOA systems, to conduct the manual searching.

5.2 Evaluation criteria

To measure the effectiveness of our approach for generating ad-hoc processes and discovering appropriate services, we use *recall* (r), *precision* (p), *top-k precision* (p_k), and *r-precision* (p_r), defined as follows.

$$p = \frac{|RetRel|}{|RetI|}, \quad r = \frac{|RetRel|}{|RelI|}, \quad p_k = \frac{|RetRel_k|}{|k|}, \quad (3)$$

$$p_r = p_{|RelI|} = \frac{|RetRel_{|RelI|}|}{|RelI|}$$

RelI is the set of relevant items (e.g., services or tasks); *RetI* is the set of returned items from a query; *RetRel* is the set of returned items that are relevant; and *RetRel_k* is the set of relevant items in the top- k returned items. *RetRel_{|RelI|}* is the number of relevant items at the top $|RelI|$ number of returned items for a query.

Precision (p) is the ratio of the number of returned relevant items to the total number of returned items. *Recall* (r) is the ratio of the number of returned relevant items to the total number of relevant items existed in the repository. However, the number of the returned items can be too large for a user to review. Instead, a user would only go through the first k returned items. 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 items. 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 precision would be $9/50=18\%$. The *R-precision* calculates the precision based on the number of relevant items at the top r returned items, and r is the total number of relevant items existed. 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 $9/20=45\%$.

5.3 Experiment procedure

5.3.1 Evaluating the generated ad-hoc processes

To evaluate if the generated ad-hoc process can match well with a user's expectation to achieve a goal, we conducted a user study. We asked the 8 subjects to manually specify 6 ad-hoc processes for 6 goals based on their experience and domain knowledge. Due to the limitation of time and recourses, we are not able to ask subjects to describe the ad-hoc processes for all the 20 goals described in Sect. 5.1.1. We randomly select 6 goals out of the 20 goals to evaluate the generated ad-hoc processes. The as-designed ad-hoc process provided by different subjects may vary a lot due

to the differentiation of their preferences and habits. For each as-designed ad-hoc process, we calculate the recall and precision based the generated ad-hoc processes and the as-designed ad-hoc processes. Given a goal, the tasks in an as-designed ad-hoc process are treated as the relevant tasks (i.e., *RelI* in formula 3), and the tasks generated by our approach are treated as returned tasks (i.e., *RetI* in formula 3). We use the average recall and precision to evaluate the overall performance of our generated ad-hoc processes.

5.3.2 Evaluating the performance of service discovery

We compare the performance of our proposed tag-based service in discovering Web services with a baseline approach which requires manually search for relevant services using keywords. Due to a large number of tasks generated in 20 ad-hoc processes, we are not able to evaluate all associated services. To compare the recall and precision of our approach with the baseline approach, we use 6 ad-hoc processes from the 20 generated ad-hoc processes. To make the *top-k precision* meaningful, each task in the 6 ad-hoc processes has at least 6 relevant services in our service repository. The novice developer manually searches for services to match with the tasks generated from the 6 ad-hoc processes.

To compare our approaches with the baseline approach using the same set of tasks, we provide the 30 discovered tasks in 6 ad-hoc processes as described in Table 3 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.

To calculate the *recall* and *r-precision*, one graduate student as one of the co-authors spent around 3 weeks in manually analyzing the 1,000 Web services registered in the service repository and to identify the relevant services for each task.

5.3.3 Evaluating user experience of the service composition

To evaluate the user satisfaction of using our approach to compose services, we conducted a user study. We ask each

Table 3 Characteristics of the six ad-hoc processes

Process ID	Name of ad-hoc processes	# of entities	# of tasks
1	Travel	47	6
2	Watching movie	28	5
3	Online shopping	29	5
4	Credit card application	24	4
5	Stock analysis	27	3
6	Job-hunting	18	7

of the 8 subjects as described in Sect. 5.3.1 to compose 3 ad-hoc processes by providing 3 goals using our prototype. After the subjects completed the service composition, we ask them to complete a short survey to assess their experience with the service composition. We indicate the rational of each question in parentheses. The survey contains the following questions:

1. Whether the relations among tasks help you to navigate among different tasks? (Easiness of task navigation)
2. Does the tag description help you understand the functionality of services and select services? (Understandability of tag-based service description)
3. How helpful is the prototype for you to find online services comparing with the other service mediator websites (e.g., Expedia)? (Helpfulness to end-users)
4. How much SOA knowledge is required for composing services using this prototype? (Requirement of SOA background).

The survey provides five choices for each question to measure the degree of the answer, such as “almost never”, “a little”, “sometimes”, “often”, and “almost always”. Those answers are mapped to the score of 0–5.5 represents the most positive answer and 0 represents the most negative answer.

5.4 Results

5.4.1 Results of evaluating the generated ad-hoc processes

The average recall and precision for the tasks in the generated ad-hoc processes are 0.80 and 0.84, respectively. We find that the as-designed ad-hoc processes vary a lot due to the differentiation of the subjects’ preferences. Therefore, the recall and precision of tasks in the generated ad-hoc processes are not very high. The result of the case study shows that the generated ad-hoc process can provide the major tasks to fulfill a user’s goal. As a complement, in our approach, end-users can edit the generated ad-hoc process to make them satisfy their own specific requirements for future reuse.

5.4.2 Results of evaluating service discovery

Table 4 lists the average recall of our approach and the baseline approach. In the searches for 30 tasks, our approaches can find all the relevant services with a recall of 93%. 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 4 lists the average r-precision of each task for both approaches. Our approach has higher r-precision than the baseline approach.

Table 4 Recall and R-precision comparison

	Recall	R-precision
Our approach	0.93	0.61
Baseline	0.67	0.33

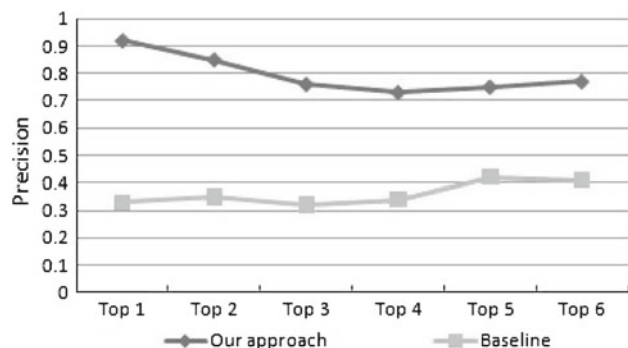


Fig. 14 Top-k precision

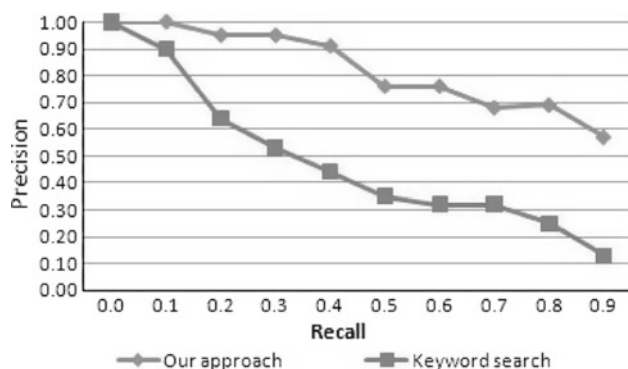


Fig. 15 Recall versus precision curves

We calculate the averages of the *top-k* precisions (ranging from *top-1* precision to *top-6* precision) for all the tasks. Figure 14 shows the results for average *top-k* precision for all 30 tasks, when *k* ranges from 1 to 6. As shown in Fig. 14, 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 Fig. 15. 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 Fig. 15. As shown in Fig. 15, 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.

Table 5 Results of satisfaction evaluation

Question no.	Metrics	Average value
1	Easiness of task navigation	4.6
2	Understandability of tag-based service description	4.4
3	Helpfulness to end-users	4.2
4	Requirement of SOA background	0.3

When searching for services, 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. The service retrieval in our approach uses the expert knowledge captured in an ontology and the tag-based services description. 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.

5.4.3 Results of evaluating user experience

Table 5 lists the average value of each question in our survey. In Table 5, the first column shows the question number corresponding to the questions in the survey. The last column is the average value of the scores provided by the 8 subjects. The results indicate that our tag-based service description can help subjects to understand the service descriptions and the generate ad-hoc process can reduce the workload of composing services. The subjects 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 them to navigate through the services. Moreover, the subjects found that the tag-based service information intuitive for understanding the functionality and usage of the services.

5.5 Threats to validity

In this section, we discuss the threats to the validity of our case studies.

External validity refers to the generalization of the results. Instead of simulating Web services and using self-designed ontologies, our case studies use the public available Web services and collect the ontologies collected from the Internet or generate the ontologies from on-line Web pages. We believe that such Web services and ontologies can better reflect the situation of the practices. However, there are a large number of Web services available on the Internet in various domains. Our case study only conducted the limited Web services from six domains. In the future, we plan to expand our services

repository and collect more services from different domains. When the number and the domains of Web service increase in our service repository, we expect that the *precision* and *recall* is likely to be a lower than the results of our experiment.

In the experiment, we note that the generated process depends on the quality of the ontology. 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 useless tasks or the generated process might miss some important tasks for achieving the goal.

In addition, our approach cannot well control the granularity of tasks in the ad-hoc processes. We find that some generated ad-hoc processes in our experiment contain too detailed tasks. Those tasks are relevant to the goal but useless to help end-users fulfill their needs. In the future, we plan to collect and analyze the context information of end-users (e.g., user's preferences and historical data). The context information might help us to select high quality ontologies and refine the ad-hoc processes.

Construct Validity is the degree to which the independent and dependent variables accurately measure the concepts which they are intended to measure. In our case studies, we have carefully chosen the criteria to avoid the threats of construct validity. To evaluate the effectiveness of generated ad-hoc processes and the tag-based service description, we use *recall* and *precision* which are the well-adopted evaluation criteria in literature. In the survey of user's satisfactions, we use multiple-choice to help participants provide their feedback accurately. However, the as-designed ad-hoc processes and user's satisfactions contain subjective issues. The results of our evaluations may not exactly represent the feedback of all the end-users in the real world.

Internal validity is a concern with the cause-effect relationship between independent and dependent variables. While comparing the baseline approach and our tag-based approach for service discovery, the tag-based approach is executed automatically using our prototype and the baseline approach is conducted by a developer who did not observe the result of the tag-based approach. Therefore, we can rule out the learning effect that the developer may learn from the generated ad-hoc processes. To avoid interference, the experiment was run with subjects who had never done a similar experiment.

Communication between the subjects influences the response of subjects. We ruled out this threat by executing the experiment in an observed room where the subjects were not allowed to communicate. In addition, subject's knowledge on the relevant domains can also impact the result. In our experiment, all the ad-hoc processes in our case studies are relevant to daily activities. Therefore, every subject can use their knowledge to provide rational evaluations on the ad-hoc processes.

6 Related work

In this section, we give an overview of the research in the areas of service description models, service composition, and techniques for supporting end-users in service composition.

6.1 Enhancing service description with semantics

Service description models specify the capabilities of services, which usually includes input/output parameters, exceptions, functional and non-functional description. WSDL is an XML-based language to specify how potential clients access a service [11]. WSDL is a well accepted industry standard. In 2007, WSDL 2.0 was endorsed and recommended by World Wide Web Consortium (W3C). WSDL describes Web services as a set of operations. An operation is an interaction between the service and the service consumer. Each interaction contains a set of messages exchanged between the service and the service consumer [11]. WSDL describes the programming interfaces of Web services. However, WSDL does not include semantic descriptions. Without semantic descriptions, service consumers and service integrators need additional background or additional documents to understand and access a service. The lack of semantic description also makes automatic services composition difficult. Moreover, WSDL is designed for programmers and is difficult to understand by non-professional end-users. In our approach, we use descriptive tags to enhance the description of WSDL, which can be easily understood by non-professional end-users.

To describe semantic meanings, various semantic Web service description models are proposed by researchers. One research trend in semantic Web services is to enhance WSDL with semantic descriptions. Semantic Annotations for WSDL and XML Schema (SAWSDL) [18] is a W3C recommendation to annotate semantics to WSDL and XML. SAWSDL provides mechanisms to associate semantic models (e.g., Ontologies) to WSDL and XML schema components. The semantic models are defined outside the WSDL document. SAWSDL does not denote any specific language for representing the semantic models. Sivashanmugam et al. [45] use the extensibility supported by WSDL specification to add semantic descriptions to WSDL. Miller et al. [3,41] create a language called WSDL-S to extend WSDL with semantic descriptions. Miller et al. assume that the semantic models already exist. The semantic models are maintained outside of WSDL documents and referenced from the WSDL document through WSDL extensibility elements. Another research trends of semantic Web services is to create a full framework for semantic Web services. Ankolekar et al. [4] use a DAML+OIL based ontology, named DAML-S, to describe the semantic meanings of Web services. OWL-S is the successor of DAML-S. OWL-S provides an OWL-based framework for describing semantic Web services. The OWL-S

ontology is written in Ontology Web Language (OWL). OWL-S uses the class “Service” to describe the knowledge about a Web service, such as what the service does, how to use the service, and how a service client can access the service. Each published Web service is mapped to an instance of “Service”. Instead of providing the concrete specification of how to access services, OWL-S uses a class named “ServiceGrounding” to construct the mapping between the semantic description of services and the concrete specification of how to access the services (e.g., WSDL). Except OWL-S, WSMO is another prominent semantic description model. WSMO defines four major components to describe semantic Web services: (1) Ontologies, which provides the terminology used by all other components; (2) Web Services, which describe the capabilities, interfaces and internal working of the Web services; (3) Goals, which represent the objectives that a client can achieve by executing the Web service; and (4) mediators, which define elements to overcome interoperability problems between different WSMO components [9].

Both research trends require formalized semantic descriptions. Due to the complexity of applying semantic descriptions for Web services in practices, there is still no completely satisfying semantic description language that is both pragmatic and formally sound [27,28,48]. In our work, instead of requiring the formal semantic descriptions from service developers, we use structured tags to enhance the descriptions of service descriptions. By simplifying the services description using tags and encouraging the participation of end-users, our approach can use the knowledge from end-users to enhance the existing service description model (i.e., WSDL) with a certain level of semantic meanings. Our approach does not increase the workload of service developers.

6.2 Service composition

Generally, we can classify the methods used for service composition into two categories: (1) Model-driven service composition; and (2) Goal-driven service composition.

Model-driven service composition uses the design data, e.g., UML (Unified Modeling Language) diagrams and workflows, as the start point, and provides approaches to transform the design data into executable business processes, such as BPEL. IBM WebSphere Business Modeler (WBM) [22] and WID are representative products of model-driven service composition. In the design stage, business analysts use WBM to capture business requirements and model business processes. Then business processes are transformed into WID as abstract BPEL processes. In the integration stage, SOA developers use WID to add details to BPEL processes, search services, and bind services to BPEL processes. To reduce the manual work, research efforts have aimed at increasing the automation of model-driven service composition

[35,36,38,39]. For example, Pistore et al. [38,39] propose an approach to automatically generate an executable BPEL process based on an abstract BPEL process, the composition requirements and the relevant services. The abstract BPEL process and a set of relevant services are automatically translated into state transition systems. The requirements of the composite service are formalized. A state transition system can be automatically generated using the state transition systems and the formalized requirements. Finally, the generated state transition system is automatically translated into an executable BPEL process. Model-driven approaches require the design data, such as UML diagrams, as the input of service composition systems. Non-professional end-users usually do not have the knowledge to provide these design data. In our approach, to enable end-users to compose services, we only ask end-users to provide a few keywords to describe their goal.

In the goal-driven approach, a goal for the service composition is expressed using either formal or informal languages. Processes are created on the fly to realize the goal. Most approaches in this category treat service composition as an Artificial Intelligence (AI) planning problem [29,42,44,52]. The goal of AI planning is to find a set of actions which can transit from the initial state to the target state. The initial state is the current state of the world; the target state is an end-user's goal. Actions are available services. For example, Wu et al. [52] use SHOP2, which is one of AI planners, to automatically compose DAML-S Web services. SHOP2 uses operators, methods, and various non-action related facts and axioms to capture the domain knowledge. A SHOP2 operator describes the inputs, preconditions, and effects of executing a primitive task. A SHOP2 method is a description on how to decompose a compound task into partially ordered subtasks. Wu et al. translate DAML-S models into the SHOP2 domains, and then use SHOP2 to compose services. The domain knowledge of SHOP2 is converted from the atomic processes and composite processes defined in DAML-S models. Sheshagiri et al. [44] present a backward-chain planner that composes services described in DAML-S into a composite service. The action (i.e., service) consists of inputs, preconditions, outputs, and effects of services. The planner starts from the final goal and repeats the following two steps using backward chain to trace the required actions (i.e., services): (1) Find services that fulfill the existing goal and save the service in a set; (2) Convert the inputs and preconditions of the services in the set into new goals, and go to step (1) until all the inputs and preconditions are satisfied or provided by the initial state. However, these approaches require semantic Web services, such as the pre and post conditions of services. The approaches have limited support for dynamic services composition when there is no formal semantic description available. Instead of requirement semantic descriptions of Web services, our approach is designed based on the exist-

ing industrial standard WSDL services which do not have formal semantic descriptions.

Similar to our approach, ontologies are used in service discovery and service composition [5,6,17]. In [5], the services are classified using ontologies to guide end-users to search for services. The approach in [6] uses ontologies to semi-automatically compose services by matching the interfaces of individual services. Different from the aforementioned approaches, our approach uses the knowledge in ontologies to dynamically identify required tasks and generates ad-hoc processes.

6.3 Supporting end-users in service composition

Mashups are browser-based applications. End-users can easily access Mashup applications using Web browsers (e.g., Internet Explorer and Firefox) without installing any software on the client side. Several products, such as Yahoo! Pipe [55] and IBM Mashup center [21], provide user friendly environment for end-users to manually connect Web resources into one Web page. Such environments are easy for non-IT professional end-users to learn and to manually compose services. However, those products require end-users to manually identify all the services to form an ad-hoc process. Our approach reduces the workload of end-users by automatically generating ad-hoc processes for end-users. Liu et al. [30] propose a Mashup architecture which extends the SOA model with Mashups to facilitate service composition. Carlson et al. [10] provide an approach to progressively compose services based on the interface matching. Given a service, Carlson et al. use the output description of the service to match the inputs of existing services in the repository. This approach can recommend the next potential services through matching the input and output descriptions. By recommending the next potential services, an end-user can compose an executable business process. Our work enhances service Mashups by providing guidance to end-users through the automatic composition of services.

In addition to Service Mashups, other approaches also provide support for end-users to compose services. The project Ubiquitous service composition for all users (i.e., UbiCompForAll) [49] provides support for non-IT professionals to compose services. UbiCompForAll conducted an initial experiment to evaluate a service composition tool which creates mobile tourist services for end-users. UbiCompForAll uses different case scenarios relevant to the city guide to develop and validate the user interfaces of the service composition tool. However, no details and concrete results on providing support for end-users have been revealed by UbiCompForAll. Obrenovic et al. [33] provide a spreadsheet-based tool to help end-user compose services. A spreadsheet (e.g., Microsoft Excel) is a software application that uses rectangular tables to display information. In a spreadsheet,

the content is specified in the cells of the table, and the relations among the cells are defined by formulas. Obrenovic et al. extend the spreadsheet to enable it to exchange messages with services and support different composition patterns. However, it is challenging for the end-users who are not familiar with the spreadsheet to understand and manipulate the data in spreadsheet, especially to use formulas to control data. Our approach generates ad-hoc processes to help end-users compose services and fulfill the goals of daily activities.

7 Conclusion and future work

In this paper, we provide an approach that hides the complexity of SOA standards and tools from end-users and semi-automatically composes services to help an end-user fulfill their daily activities. We propose a tag-based service description to allow end-users to understand the description of a service and add their own descriptive tags. An ad-hoc process model is presented to describe the loose coupled task relations for fulfilling the goals of daily activities. Using our approach, an end-user only needs to specify the goal of their activities using keywords. Our approach can compose services that help an end-user achieve their desired goals without requiring the user to specify the detailed tasks. Our case studies evaluate the quality of generated ad-hoc processes, the precision and recall of our tag-based service description, and user's satisfactions. The results of the case studies demonstrate the effectiveness of our approach.

To generate better ad-hoc processes, it might be helpful if we can find and merge the information from more than one ontology especially when the keywords in the goal description are unrelated. In the next step, we plan to enhance our current approach by considering using the information from more than one ontology to generate ad-hoc processes. In addition, we plan to conduct a large scale case study to better evaluate our approach.

Acknowledgments This work is financially supported by the IBM Toronto Centre for Advanced Studies and NSERC. IBM and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. Other company, product, and service names may be trademarks or service marks of others.

References

1. ActiveBPEL Designer Sheet (2010) Available at: http://cmapps.public.ihmc.us/rid=1172082594921_158601157_643/activebpel_designer_datasheet.pdf. Last time Accessed on 17 May 2010
2. Aguilar-Lopez D, Lopez-Arevalo I, Sosa-Sosa V (2008) Toward the semantic search by using ontologies. In: International conference on electrical engineering, computing science and automatic control, Mexico City, Mexico, Nov 2008, pp 328–333
3. Akkiraju R, Arrell J, Miller J et al (2005) Web service semantics-WSDL-S. W3C member submission, 7 Nov 2005
4. Ankolekar A et al (2002) DAML-S: web service description for the semantic web. In: International semantic web conference, Sardinia, Italy, June 2002, pp 348–363
5. Arabshian K, Dickmann C, Schulzrinne H (2009) Ontology-based service discovery front-end interface for GloServ. In: LNCS in the semantic web: research and applications, 2009, pp 684–696
6. Arpinar IB, Aleman-Meza B, Zhang R, Maduko A (2004) Ontology-driven web services composition platform. In: IEEE International conference on e-commerce technology, San Diego, July 2004, pp 146–152
7. Beckett D, McBride B (2004) RDF/XML syntax specification (Revised), W3C Recommendation, 2004
8. Brin S, Page L (1998) The anatomy of a large-scale hypertextual web search engine. In: International conference on world wide web, Brisbane, Australia, Apr 1998, pp 107–117
9. Bruijn JD et al (2010) Web service modeling ontology (WSMO). WSMO final draft 21, Oct 2006. Available at: <http://www.wsmo.org/TR/d2/v1.3/>. Last time Accessed on 14 May 2010
10. Carlson MP, Ngu AHH, Podorozhny RM, Zeng L (2008) Automatic mash up of composite applications. In: International conference on service oriented computing, Sydney, Ultimo City Campus, Dec 2008, pp 317–330
11. Chinnici R, Moreau J, Ryman A, Weerawarana S (2007) Web service description language version 2.0. W3C Recommendation, 2007
12. Cimiano P, Volker J (2005) Text2Onto—a framework for ontology learning and data-driven change discovery. In: Proceedings of 10th international conference on applications of natural language to information system (NLDB) 2005, Alicante, Spain, June 2005, pp 227–238
13. Connolly D, Harmelen FV, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2010) DAML+OIL reference description. W3C Note, Dec 2001. Available at: <http://www.w3.org/TR/daml+oil-reference>. Last time Accessed on 17 May 2010
14. DBpedia (2010) Available at: <http://dbpedia.org/About>. Last Accessed on 18 May 2010
15. Expedia (2010) Available at: <http://www.expedia.com/default.asp>. Last Visited on 18 May 2010
16. Facebook (2010) <http://www.facebook.com/>. Last Accessed on 14 May 2010
17. Fang W, Zhang L, Wang Y, Dong S (2005) Toward a semantic search engine based on ontologies. In: International conference on machine learning and cybernetics, Guangzhou, China, Apr 2005, pp 1913–1918
18. Farrell J, Lausen H (2007) Semantic annotation for WSDL and XML schema. W3C Recommendation, 28 Aug 2007
19. Freebase (2010) Available at: <http://www.freebase.com/>. Last Accessed on 18 May 2010
20. Google (2010) Available at: <http://www.google.com>. Last Accessed on 18 May 2010
21. IBM (2010) Mashup center, <http://www-01.ibm.com/software/info/mashup-center/>. Last Accessed on 18 May 2010
22. IBM WebSphere Business Modeler (2010) Available at: <http://www-01.ibm.com/software/integration/wbimodeler/advanced/features/>. Last Accessed on 18 May 2010
23. IBM WebSphere Integration Developer (2010) <http://www-01.ibm.com/software/integration/wid/>. Last Access on 18 May 2010
24. IBM WebSphere Service Registry and Repository (2010) <http://www-01.ibm.com/software/integration/wsrr/>. Last Access on 18 May 2010
25. Jordan D et al (2010) Web services business process execution language. OASIS standard, Apr 11 2007. Available at:

- <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>. Last Accessed on 16 May 2010
26. Klyne G, Carroll JJ, McBride B (2004) Resource description framework (RDF): concepts and abstract syntax. W3C Recommendation, Feb 10 2004
 27. Klusch M, Xing Z (2007) Semantic web service in the web: a preliminary reality check. First international joint ISWC workshop SMR2 2007 on service matchmaking and resource retrieval in the semantic web, Busan, Korea, 2007
 28. Kuroopka D, Meyer H, Troger P, Weske M (2008) Application and outlook. In: Book semantic service provisioning, published by Springer Berlin Heidelberg, 2008, pp 203–210
 29. Küster U, Stern M, König-Ries B (2005) A classification of issues and approaches in service composition. International Workshop on Engineering Service Compositions, Amsterdam, The Netherlands, Dec 2005
 30. Liu X, Hui Y, Sun W, Liang H (2007) Towards service composition based on Mashup. In: IEEE congress on services, 2007
 31. Martin D et al (2004) OWL-S: semantic markup for web services. Technical Report, Member Submission, W3C, 2004
 32. Nielsen J (2001) Success rate: the simplest usability metrics. Jakob Nielsen's Alertbox, Feb 2001
 33. Obrenovic Z, Gasevic D (2008) End-user service composition: spreadsheets as a service composition tool. IEEE Trans Serv Comput 1(4), October–December 2008
 34. Oracle BPEL Process Manager (2010) Available at: <http://www.oracle.com/technology/products/ias/bpel/index.html>. Last Accessed on 17 May 2010
 35. Orriüens B, Yang J, Papazoglou MP (2003) A framework for business rule driven web service composition. In: ER 2003 workshops, LNCS 2814, Springer, Berlin, Heidelberg, 2003, pp 52–64
 36. Orriüens B, Yang J, Papazoglou MP (2003) Model driven service composition. In: International conference on service-oriented computing (ICSOC) 2003, Trento, Italy, Dec 15–18 2003, pp 75–90
 37. Paolucci M, Kawamura T (2002) Semantic matching of web services capabilities. In: International semantic web conference (ISWC) 2002, Sardinia, Italy, June 10–12 2002, pp 333–347
 38. Pistore M, Marconi A, Bertoli P, Traverso P (2005) Automated composition of web services by planning at the knowledge level. In: International joint conference on artificial intelligence (IJCAI) 2005, Pasadena, California, USA, pp 1252–1259
 39. Pistore M, Traverso P, Bertoli P, Marconi A (2005) Automated synthesis of composite BPEL4WS web services. In: International conference on web services (ICWS) 2005, Orlando Florida, USA, July 11–15, 2005, pp 293–301
 40. Protégé (2011) <http://protege.stanford.edu/>. Last time Accessed on 20 Feb 2011
 41. Rajasekaran P, Miller J, Verma K, Sheth A (2004) Enhancing web services description and discovery to facilitate composition. In: International workshop on semantic web services and web process composition, San Diego, CA, USA, July 2004
 42. Rao J, Su X (2004) A survey of automated web service composition methods. In: International workshop on semantic web services and web process composition, San Diego, CA, USA, July 2004
 43. Seekda (2010) <http://webservices.seekda.com/>. Last Accessed on 18 May 2010
 44. Sheshagiri M, DesJardins M, Finin T (2003) A planner for composing services described in DAML-S. AAMAS Workshop on Web Services and Agent-Based Engineering Melbourne, Australia, July 2003
 45. Sivashanmugam K, Miller JA, Sheth A, Verma K (2004) Framework for semantic web process composition. Int J Electron Commer 9(2):71–106
 46. Smith MK, Welty C, McGuinness DL (2004) OWL web ontology language guide. W3C Recommendation, 2004
 47. Swoogle (2010) <http://swoogle.umbc.edu/>. Last Accessed on 16 May 2010
 48. Torma S, Villstedt J, Lehtinen V, Oliver I, Luukkala V (2008) Semantic web services—a survey. Technical Report published by Laboratory of Software Technology, Helsinki University of Technology, Mar 2008
 49. UbiCompForAll—Ubiquitous service composition for all users (2011) <http://www.sintef.no/Projectweb/UbiCompForAll/Home/>. Last Accessed on 20 Feb 2011
 50. Wikipedia (2010) Available at: <http://www.wikipedia.org/>. Last Accessed on 18 May 2010
 51. WordNet (2010) Available at <http://wordnet.princeton.edu/>. Last time Accessed on 17 May 2010
 52. Wu D, Parsia B, Sirin E, Hendler J, Nau D (2003) Automating DAML-S web services composition using SHOP2. In: International semantic web conference, Sanibel Island, Florida, USA, Oct 2003, pp 195–210
 53. Xiao H, Zou Y, Tang R, Ng J, Nigul L (2010) A framework for automatically supporting end-users in service composition. In: The smart internet, Lecture Notes in Computer Science, vol 6400/2010, Springer, pp 115–136
 54. Xiao H, Zou Y, Tang R, Ng J, Nigul L (2009) An automatic approach for ontology-driven service composition. In: IEEE international conference on service-oriented computing and applications (SOCA) 2009, Taipei, Taiwan, Dec 2009, pp 1–8
 55. Yahoo! Pipes (2011) <http://pipes.yahoo.com/pipes/>. Last Accessed on 20 Feb 2011