A Framework for Automatically Supporting End-Users in Service Composition

Hua Xiao[1], Ying Zou[2], Ran Tang[2], Joanna Ng[3], Leho Nigul[3]

[1] School of Computing, Queen's University
Kingston, Ontario, Canada
huaxiao@cs.queensu.ca

[2] Dept. Of Electrical and Computer Engineering, Queen's University
Kingston, Ontorio, Canada
{ying.zou, ran.tang}@queensu.ca

[3] IBM Toronto Lab, Markham, Ontario, Canada
{jwng, lnigul}@ca.ibm.com

Abstract. In Service Oriented Architecture (SOA), service composition integrates existing services to fulfill specific tasks using a set of standards and tools. However, current service composition techniques and tools are mainly designed for SOA professionals. It becomes challenging for end-users without sufficient service composition skills to compose services. In this paper, we propose a framework that supports end-users to dynamically compose and personalize services to meet their own context. Instead of requiring end-users to specify detailed steps in the composition, our framework only requires the end-users specify the goals of their desired activities using a few keywords to generate a task list. To organize the task list, we analyze the historical usage data and recover the control flows among the tasks in the task list. We also mine the task usage pattern from the historical usage data to recommend new services. A prototype is designed and developed as a proof of concept to demonstrate that our approach enables end-users to discover and compose services easily.

Keywords: Personalized Service Composition, Context-awareness, Mining Historical Usage Data, Service Oriented Architecture

1 Introduction

Web services are considered as self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. The Service-Oriented Architecture (SOA) uses loosely coupled Web services as basic constructs to build more complex systems in a flexible and rapid way. In particular, single service generally cannot fulfill the functionality required by end-users. A significant amount of effort from industry and academics intends to provide infrastructure, languages and tools to compose the services using well-defined business processes to streamline business operations. Such SOA systems tackle complex business requirements across organizations. However, little attention has been paid to allowing end-users to compose services to fulfill their daily activities that can be represented as ad-hoc processes. For example, planning a trip is an ad-hoc process for many end-users. It involves several tasks, such as searching for transportation, booking accommodation, and checking restaurants. These tasks can be performed dynamically to achieve the goal of trip planning.

In today's on-line experience, an end-user, who is not familiar with Web service standards and tools, frequently re-visits websites and uses on-line services to perform daily activities, such as planning a trip. 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 pre-defined plan. Consider a scenario that a person plans a trip to Vancouver in two contexts: (1) John, a business man, lives in Toronto and travels to Vancouver for a business meeting. He would like to take Air Canada, as a reliable transportation vehicle, to stay in the Marriott since he is a valued member of Marriott, and to find a formal seafood restaurant for an important business dinner. (2) Emily, who is a student at the University of Calgary, wants to take a vacation in Vancouver. She chooses to plan her trip at the lowest cost. She will take Greyhound bus, search for a motel in Vancouver and look for fast food chains, such as Wendy's and MacDonald's. In both contexts, each end-user has to manually browse different services offered by Websites and Web services to gather information for performing the same tasks in the trip planning. It is often challenging for end-users to know all the relevant Web services or websites that help them to make a decision.

Specialized service mediators, such as expedia.com [13], can be used to search for accommodation, flight and train information. However, it cannot automatically provide services that are personalized to a particular end-user (e.g., traveler with luxury plans, or traveler on a small budget). Moreover, the services provided by the service mediators are pre-defined and limited. For example, the end-user has to visit other websites to check the reviews about a hotel or a restaurant and check if the airline suggested by expedia.com is in part of the same alliance so they can collect their frequent flyer points. The end-user has to repeat the same step when planning the next trip. It would be ideal if an end-user can compose a travel planning service using the SOA techniques. Then the composed service automatically gathers the needed information and presents it to the end-user based on their preference.

In the current state of practice, composing services 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 composed service, the professionals in different roles and tools must interact in harmony. Unfortunately, non-expert end-users do not possess knowledge of most of these tools and lack the knowledge of SOA standards. In short, involving end-users in service composition has the following two challenges:

- **Complexity in service description and discovery.** The 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. However, WSDL is too complex for non-expert end-users. WSDL is primarily intended for SOA experts to understand the interface and parameters of a Web service and to correctly invoke a service. It creates difficulty for end-users to understand the functionality of a service to discover an appropriate service.

- **Limited support for composing services on the fly.** A system integrator can specify BPEL (Business Process Execution Language) [38] processes to compose Web services using tools, such as IBM WID (WebSphere Integration Developer) [19]. After deployment of a Web service, 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. An ad-hoc process involves the dynamic integration of various services (e.g., Web services, and websites) on-the-fly. Moreover, it is also infeasible to expect an end-user to specify the details of each task and orchestrate a well-defined process in BPEL.

These challenges represent barriers for integrating SOA into an end-user's activities for improved quality of life and productivity. To address these challenges, our work focuses on the following two aspects:

1) **Context aware service discovery.** We design and develop techniques that automatically capture an end-user's context which denotes changes in an end-user's environment (i.e., operational and physical environment), and use the context to assist in service discovery. This avoids the need for end-users to understand the WSDL description of a Web service. It would also reduce the amount of information required from end-users in order to specify well-defined criteria for service discovery.
2) **Dynamic service composition.** We devise techniques that dynamically search and compose services on-the-fly to assist an end-user in fulfilling their desired goals (such as planning a trip). We aim to provide an approach to shelter the end-user from complex programming issues.

The remainder of this paper is organized as follows. Section 2 discusses the requirements and potential techniques for enabling end-user friendly service composition. Section 3 presents our framework that automatically composes services on the fly and personalizes composition results for end-users. Section 4 describes a proof of concept prototype that implements the proposed framework. Section 5 gives an overview of related work. Finally, section 6 concludes the paper and presents the future work.

2 Requirements and Techniques for Building End-User Friendly Service Composition Environments

It is challenging for end-users without deep understanding of SOA knowledge and standards to integrate SOA technology in their daily on-line experience. In this section, we discuss the requirements crucial for the involvement of end-users in the service composition process. Then we introduce the enabling techniques that support the achievement of the requirements.

2.1 Requirements for End-User Friendly Service Composition Environments

Generally, end-users are most concerned with the ease of use of service composition tools. The tools are expected to be intuitive to use without a steep learning curve, be quickly mastered, and simplify their daily tasks. The technology details (e.g., how the services are developed, deployed and delivered) is not the end-user's concern. To achieve end-user's expectation, we envision that the following requirements are important to acquire in the service composition environment.

(1) Automatic Generation of Ad-Hoc Processes

In the current state of practice, SOA practitioners manually describe the details of each task and the interactions among tasks using BPEL. In an end-user environment, most of the activities or goals (e.g., plan a trip) are spontaneously prompted from an end-user. An end-user may not have a clear plan on achieving a goal. To guide an end-user to achieve their goal, our aim is to reduce the amount of inputs required from end-users. Instead of specifying the complete tasks to achieve their goal, an end-user is required to describe a high level goal using keywords. For example, for planning a trip, an end-user can specify the keywords, such as travel, Vancouver and student. To understand the semantic meaning of an end-user's goal, ontologies provide rich and machine-understandable semantic meanings for a given goal. We use ontologies to expand the semantic meanings of the keywords and automatically infer the possible tasks that need to be performed by an end-user. To assist end-users to walk through the task list, we track the usage of tasks by the end-users of the same goal. We identify the possible execution orders among tasks performed by the majority of the end-users to guide end-users to complete the tasks in the list in a particular order. An ad-hoc process contains a list of tasks and the suggested execution orders among the tasks.

(2) Context aware service discovery

Effective service discovery requires the ability to detect an end-user's context at the run-time environment. This context characterizes the operational conditions for an end-user to fulfill a goal, such as server workloads, end-user's profiles, computation resources, scheduled tasks, and end-user's preferences. The context of the run-time environment should also capture the functional and non-functional capabilities of deployed services. For example, non-functional requirements, such as processing time, end-user's ranking and provider reputation, can be taken into account during the discovery of deployed services. We develop techniques for automatically capturing and using the context to assist in generating searching criteria and providing personalized service discovery. This would reduce the amount of information required from end-users in order to specify well-defined criteria for searching Web services.

(3) Accurate service recommendation

A major challenge when composing services is to predicate the needed services. Current SOA runtime environment, service explorer tools allow a system integrator to search for services from the service repository. However, the results returned by the explorer are usually context insensitive. To improve the accuracy in recommending services, we trace the execution of the already generated task list and recover the usage patterns among tasks. Each task is associated with a set of already discovered services. We recommend a desired service by retrieving the task usage patterns of historical task lists that have the similar context as the desired service. For example, if an end-user is developing a service for a bookstore and other end-users already have a task list to produce a service for a video store deployed, then we can use the task list for the video store which contains services such as the inventory management, account management, credit card authorization to improve the search results. The end-user who is developing the bookstore may have already performed tasks such as picking an inventory management and selecting account management service, so we can recommend a credit card authorization task based on the knowledge learned from other task list.

Using data mining techniques (such as frequent item set mining), we can give estimate on the accuracy of our recommendation in the same manner that online bookstores such as amazon.com give for their book recommendation. For example, we can recommend a credit card authorization service and indicate that the recommended service is used by 90% of the services which have an inventory management service and account management service. Recommending services would reduce the complexity of composing services and would help an end-user pick the most popular (and ideally most widely deployed and tested) services to build their new services. This work would in turn permit the creation of high-quality well-tested services.

(4) Customizable composition results

The generated ad-hoc process guides end-users to invoke Web services and achieve their goal. However, a Web service in the generated ad-hoc process may not provide all the functionality that an end-user needs. For the example of the trip planning, an end-user may visit trip advisor website (a HTML based website) before making decision. The Web services need to interoperate with other Web resources. A Mashup is a Web page or application that remixes data or functionalities from two or more external sources to create a new service. In our work, Mashup provides a light-weight user-friendly graphic environment for an end-user to customize the composed services as well as integrate various Web resources and Web services.

2.2 Enabling Techniques

2.2.1 Structure of Ontologies

An ontology expresses common concepts (e.g., people, travel and weather), and the relations among the concepts. Fig. 1 illustrates an example ontology that defines the concept “travel”. The semantic of a high level concept (e.g., “travel”) can be further expanded into four sub-concepts: “Transportation”, “Accommodation”, “Tourist Attraction” and “Car Rental”.

Ontologies are manually described using different ontology specification languages, such as Web Ontology Language (OWL) [34] and Resource Definition

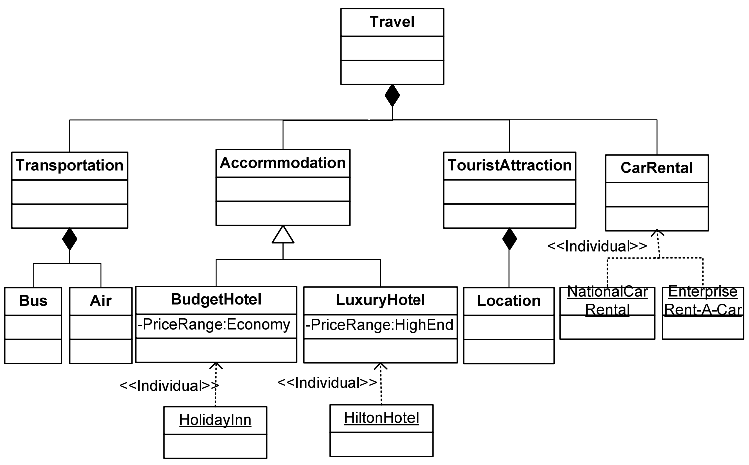


Fig. 1. An example ontology

Framework (RDF) [7]. To capture the general structure and concepts of ontologies specified in various languages, we summarize the common entities and relations in different ontologies as follows.

- *Class*: is an abstract description of a group of resources with similar characteristics. For example, "LuxuryHotel" is a *class* which describes the common characteristics of different hotels. *Class* is also called "concept", "type", "category" and "kind" in various ontology languages.
- *Individual*: is an instance of a *class*. For example, "Hilton Hotel" is an *instance* of *class* "Hotel", i.e., "Hilton Hotel" is an *individual*.
- *Relation*: defines ways in which *classes* and *individuals* can be related to one another. Typical *relations* include *subclass*, *partOf*, *complement*, *intersection* and *equivalent*. *Subclass* extends a *class* to convey more concrete knowledge. *PartOf* relation indicates that a class is a part of another class. *Complement* selects all members (classes or individuals) from the domain of the ontology that do not belong to a certain class. *Intersection* describes a class which contains the members shared by all the classes in the given class list. *Equivalent* expresses that two classes contains exactly the same set of individuals.
- *Attribute*: describes a property of a *class* (and an *individual*). *Attribute* is also called "aspect", "property", "feature" or "characteristic" in various ontology specification languages. For example, "PriceRange" is an *attribute* of *class* "LuxuryHotel".

2.2.2 Contexts and context-aware systems

A context involves several pairs of context types and context values. Specifically, a context type describes a characteristic of the context. A context type is associated with a specific context value which may vary over time with the changing environment. For example, the context types for an end-user could include location, identity, and time. The context type "location" may be assigned with a value, such as "New York". A context scenario is the combination of different context types with specific context values to reflect an end-user's situation. A context model is used to specify the relations and storage structure of various context types and values.

A context-aware system is designed to adapt the operations of systems to an end-user's context without the explicit intervention from the user [5]. A context aware system generally consists of two parts: (1) sensing and managing contexts; and (2) adapting the system to the changing contexts. Context sensors are used to retrieve context values from different resources. The Global Positioning System (GPS) receiver is an example context sensor to detect the location. To adapt the behaviors based on different context scenarios, the most common approach for context-aware systems is to manually construct mapping rules between the context scenarios and the corresponding reactions. For example, IF-THEN rules are commonly used to the mapping rules. More specially, the IF sentence describes the context scenario; and the THEN sentence represents the corresponding reactions. For example, if a person is in a meeting and receiving a phone call, then his/her cell phone is automatically switched from the ringing mode to silence or vibrate mode.

2.2.3 Mashups

A Mashup is a lightweight Web application that integrates multiple data or functions into one application to create a new service. Mashups are generally browser-based applications. End-users can easily access the Mashup applications using Web browsers (e.g., Internet Explorer and Firefox) without installing any software on the client side. Several products, such as Microsoft Popfly[27], Yahoo! Pipe [42] and IBM Mashup center[18], provide user friendly environment for end-users to manually connect Web resources into one Web page. Such environments are easier for non-expert end-users to learn and to manually compose services. For example, an end-user can integrate email client, Google calendar and weather report into a single Web page by connecting data flows among Web resources and dropping a new Web resource into an existing page.

3 An Overview of our Framework

Fig. 2 provides an overview of our proposed framework for generating and personalizing ad-hoc processes. The framework is built using client/server architecture. On the client side, end-users interact with the service composition environment through a composition user interface (UI) which is built using Mashup pages. The composition UI provides an end-user interface to enable end-users to specify the goal, navigate through the generated ad-hoc process, edit the process, and select services. To capture an end-user's context, context sensors are developed to monitor the end-user's activities in their computing environment. We deploy context sensors as plug-ins into various applications, such as Web browsers and on-line calendars.

The server contains three major components that process the contextual information gathered from the client, generate an ad-hoc process and analyze historic usage data to recommend services. The components are described as follows:

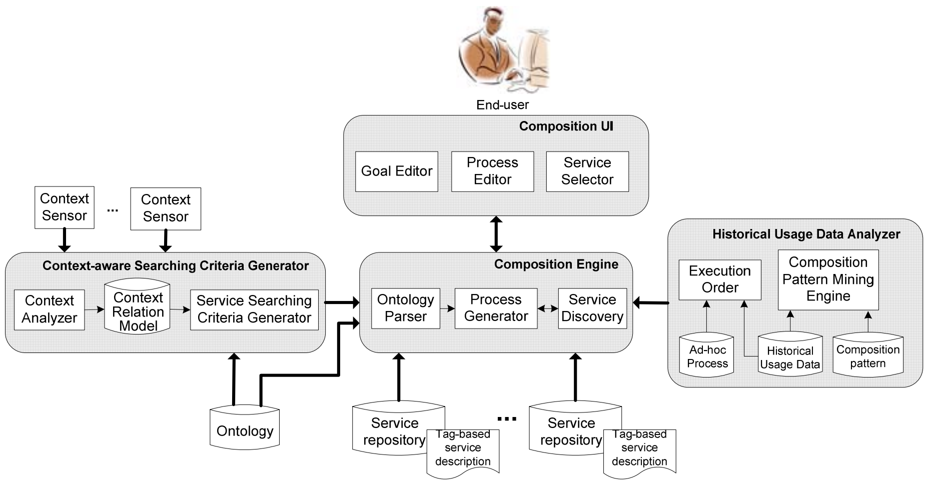


Fig. 2. An overview of our framework

Composition engine receives the goal of service composition from end-users and automatically composes a personalized ad-hoc process. The composition engine uses the goal description (i.e., keywords) to find a matching ontology. In an ontology, the semantic of a high-level goal is expanded into more concrete concepts. The composition engine uses the concepts as keywords to search services from service repositories. Service repository allows service providers to advertise their services and provide interfaces for automatic service discovery. To facilitate the execution and selection of the services, we abstract the discovered services into tasks and use historical usage data to identify the control structures among tasks. The resulting ad-hoc process can be stored and shared among multiple end-users. To enable end-users and composition tools to understand the properties of Web services, we describe services using descriptive tags (i.e., keywords). The detailed information of the tag-based service description schema is described in our earlier publication [41].

Context-aware searching criteria generator captures and analyzes the changing context scenarios of an end-user. This component automatically formulates searching criteria to discover the desired services in the service repositories. To formulate searching criteria to meet contextual requirements, existing approaches pre-define an end-user's context using context models in various forms [35]. Rules are generally specified to connect the context types to the possible end-user's preferences and generate searching criteria for service discovery. However, the context types may vary considerably between end-users. It is challenging to anticipate a complete set of context types to satisfy all the possible end-users. Fixed rules are not flexible enough to accommodate the changing environment and various personal interests. Different from existing approaches which require end-users manually specify the relations among context types, we seek an automatic approach to recognizing the relations without overwhelming end-users. For example, luxury hotel and limited budget are two context types in conflict. Therefore, the services for booking luxury hotels are automatically filtered when an end-user has very limited budget. We expect that such relations can be used to express more accurate searching criteria to reflect an end-user's context.

Historical usage data analyzer analyses the historical usage data to mine execution order among the tasks in the task list. Moreover, the historical usage data analyzer identifies frequent task usage patterns from historical usage data. To obtain the historical usage data, we track end-users' interaction with the server, such as the selected tasks, the invoked services and the time of the invocation. When an end-user executes a task list, the executed tasks are recorded as an execution instance. In each execution instance, the tasks are ordered in which they are executed. When many end-users have executed a task list, a large amount of execution instances are collected. These execution instances can be used to mine execution order of the underlying task list. The historical usage data are generally analyzed periodically to recognize task usage patterns and store them in a database. When a task list is composed, a new task can be recommended by matching the task list with the stored patterns. Each task is associated with a set of already discovered services. Consequently the associated services are recommended to the end-user.

In the following sub-sections, we present the details of the three major components: service composition engine, context-aware searching criteria generator, and historical usage data analyzer.

3.1 Ontology-based Ad-Hoc Process Generation

Once an end-user specifies the goal using keywords, we find the relevant ontology to extend the semantic meanings of those keywords. The concept (i.e., classes, individuals, and attributes) defined in the ontology could be used to search for relevant services. However, a large number of services could be returned and mixed together. It is a tedious job for end-users to manually organize and select their desired services. To guide the execution of services, we propose an approach to sort the functionally similar services under one task and identify the execution order of tasks based on historical usage data. Specifically, we use the relations of concepts in the ontology to identify a list of unique tasks which are associated with one or more functionally similar returned services. Then, we mine the task execution order using historical usage data. Finally, we can generate an ad-hoc process which contains a set of tasks with suggested execution order.

To identify tasks, we decompose the goal using the ontology to locate a subset of concepts used for service discovery. We use a depth first traversal to find the most concrete class or individual in the search criteria. A terminal node, such as "*location*", "*BudgetHotel*" and "*LuxuryHotel*" shown in Fig. 1, is a terminal node in the ontology graph which does not have child classes or individuals. The attributes of classes or individuals are included as a part of the node in the ontology graph. Terminal nodes represent the most detailed knowledge about the root class which is the goal specified by end-users. Therefore, we visit the terminal node which has the longest path from the root concept. Simply using a terminal node in the search criteria may prevent us from discovering some services since a single keyword from the terminal node provides limited knowledge. Similar to some search engines, which use the expanded query to search for the relevant documents [8], we extend the search keywords by

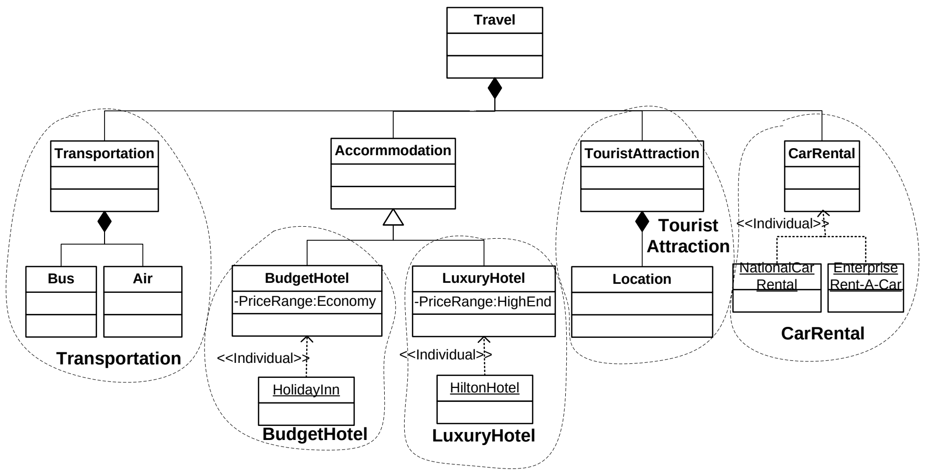


Fig. 3. An example of generating a task list

including the parent node, the attributes of the terminal node, and the sibling terminal nodes as the keywords to search for services in service repositories. For example, as shown in Fig. 3, *"Holiday Inn"* is a terminal node with the longest paths from the root "*Travel*". Its parent, *"BudgetHotel"* and attribute, *"PriceRange: Economy"*, are collected as keywords to search for services (i.e., *keywords(Budget Hotel)={Budget Hotel, Holiday Inn, Price Range, economy}*). Once a service is identified from the group of classes, individuals and attributes, these concepts are marked as a task, shown in Fig. 3. We derive the task name from the name of the parent class. For the example shown in Fig. 3, the task corresponding to individual *"Holiday Inn"* and class *"Budget Hotel"* is named after the parent, *"Budget Hotel"*. If no relevant Web services are retrieved from the path, we remove the failed nodes (i.e., the parent along the children) from the ontology graph. We recursively identify the next terminal node with the longest path from the root and repeat the same procedure, until all the terminal nodes are visited. Finally, we get a list of tasks, and each task maps to several services with similar functionality.

To help end-users navigate and execute the tasks, we mine the task execution order using historical usage data. A task list contains many tasks. The execution order between the tasks may be found in the historical usage data. Historical usage data comprises of execution instances. Each execution instance may contain execution order of some tasks in the task list. Suppose we have a task list {A, B, C, D, E}. It is possible that one execution instance in the historical usage data has executed tasks A and B, and another execution instance has executed tasks C, D and E. Thus, the execution order between tasks A and B can be found in one execution instance and the execution order among tasks C, D and E can be found in another execution instance. By aggregating all execution instances, we can obtain the overall execution order among the task list {A, B, C, D, E}. Specifically, we identify the execution order using the following steps:

1) Find the execution instances that record the execution of more than one task in the task list in the historical usage data.

2) Identify the execution flow among each execution instance which is a graph where the nodes are tasks and the edges represent execution order between tasks. The edges are determined using the ordering of tasks in the execution instance. If task A immediately precedes task B in the execution instance, we create a direct edge from A to B in the execution flow. Loops can exist in the execution flow if some tasks are repeated in the execution instance.
3) Combine the execution flow of each execution instance to obtain an overall execution flow of the task list. The overall execution flow contains all tasks in the task list as nodes. The edges from execution instances are merged in the overall execution flow.
4) Recognize different control flow structures (e.g., sequence, alternative, parallel and loop) to describe the task execution order in the task list. A sequence structure organizes tasks in a sequential order. Parallel structure and alternative structure contain multiple execution paths. In a parallel structure, all paths need to be executed. In an alternative structure, only one path can be executed. A loop structure contains tasks that can be repeated several times in one execution instance. To identify the control flow structures, we check all the execution instances to locate a starting task (i.e., the first task to be executed) and traverse the overall execution graph from the starting task. If a loop edge is encountered during the traversal, we mark the repeated tasks in a loop structure. When several paths branch out from one task and the tasks on the different paths are all executed without a particular order, we treat such paths as a parallel structure. If several paths branch out from one task and the tasks on the different paths cannot be executed in the same execution instance, we convert such paths as an alternative structure.

As an example shown in Fig. 4, we identify the execution order of the task list generated from the ontology shown in Fig. 3 to form an ad-hoc process which connects a set of tasks with control flow constructs. However, the historical usage data may not always available. When there is no historical usage data available, we use the knowledge from the ontology to identify some simple relations (i.e., alternative and parallel relations) among tasks. The detailed approach of identifying relations among tasks using ontologies are described in our earlier publication [41].

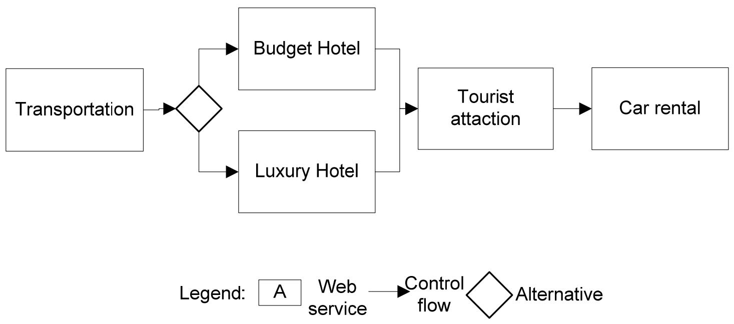


Fig. 4. An example ad-hoc process after the execution order is identified

3.2 Generating Context Aware Searching Criteria

In an end-user's context, context types can be dynamically added and removed due to the changing environment. The value of a context type also evolves over time as an end-user's computing environment changes. We need a flexible context model to dynamically adjust the context relations to accommodate the updated context types and their values. Context values capture more accurate characteristics of an end-user's context than a context type. For example, location is a context type. When the location changes, the GPS device can identify and update the location with a concrete context value, such as "Los Angeles". To correctly model an end-user's context, we focus on capturing the relations among context values instead of the high-level relations among context types. Moreover, we analyze the relations among context values to derive the searching criteria for the desired services that match the context.

Identify relations among context values. To correctly model the relations among context values, it is critical to understand the semantic meanings of each context value. Ontologies capture the relevant information related to a particular concept using expert knowledge. To gain a better understanding of a context value, we search for existing ontologies that describe the meaning of a context value. To derive the relations among multiple context values, we can check the overlaps of concepts in the corresponding ontologies.

We identify three types of relations among two context values: intersection, complement and equivalence. Intersection relation refers to the case that the ontologies of two context values contain common entities (e.g., classes or individuals). Fig. 5 illustrates examples of intersection relations. In Fig. 5, the context values "*travel*" (i.e., the context type is "activity") and "*Los Angeles*" (i.e., the context

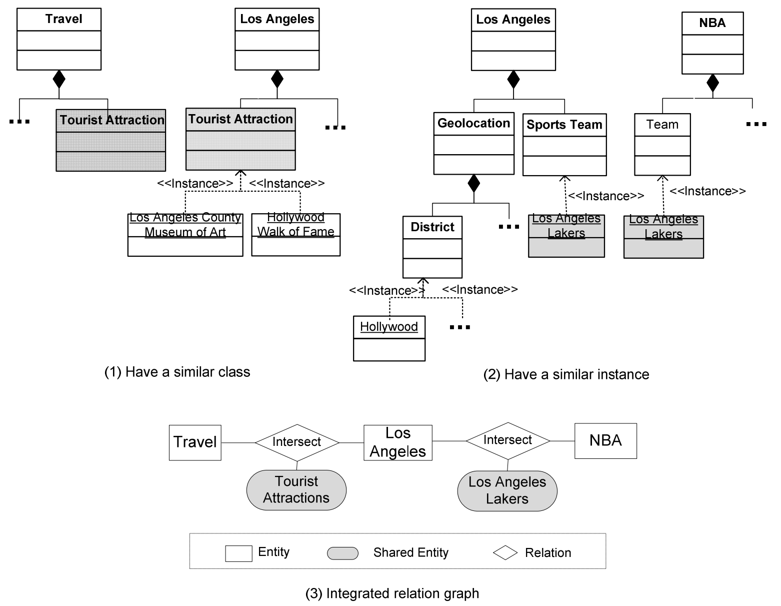


Fig. 5. Example of relations between two context values

type is "location") contain the common class "*Tourist Attraction*". Context values "*Los Angeles*" and "*NBA*" contain the same class "*Los Angeles Lakers*". When the ontologies of two context values have no overlaps and these two context values together represent all the information in the given domain, we say that such context values complement each other. For example, "*Budget Hotel*" and "*Luxury Hotel*" are in a complement relation. The equivalent relation means that the ontologies of two context values are the same. Equivalent relations are usually explicitly specified in the ontologies of the context values. The relations between two context values are inherited from the relations between their corresponding ontologies. Furthermore, we combine the relations between two context values to construct a relation graph that describes the relations among all context values. The context values with the equivalent relations are merged into one in the relation graph. An example of the relation graph is shown in Fig. 5(3). As a result, the intersection and complement relations among context values are captured in a relation graph.

Generate service searching criteria. Service searching criteria are a set of keywords which describe the potential services that an end-user is interested in a given context scenario. In our approach, service searching criteria are generated from the entities in the intersection of the ontologies of the context values. For example shown in Fig. 5, an end-user travels to "*Los Angeles*", and likes NBA games. The ontologies of "*Los Angeles*" and "*NBA*" illustrate an intersection relation between "*Los Angeles*" and "*NBA*" (i.e., the common entity is "*Los Angeles Lakers*"). It is likely that the end-user would be interested in the game played by "*Los Angeles Lakers*". To generate searching criteria, we analyze the intersection of ontologies to extract the name, the attributes and instances of the entities in the intersection. The entities in an intersection relations are expressed as AND relations in the searching criteria. The complement relations among context values reflect the conflicting context values. We use the complement relations to determine the alternative expressions (i.e., OR) in the generated searching criteria. For example, an end-user prefers to "*Budget Hotel*" as specified in the end-user profile. However, the end-user always takes "*Luxury Hotel*" by analyzing historical usage data. Both "*Budget Hotel*" and "*Luxury Hotel*" have an intersection relation with the context value "*Travel*". Generally, the "*Budget Hotel*" and "*Luxury Hotel*" cannot be reserved in the same time due to the complement relation between "*Budget Hotel*" and "*Luxury Hotel*". In the searching criteria, "*Budget Hotel*" and "*Luxury Hotel*" are expressed as an alternative relation.

If services can be discovered using the generated searching criteria, a new task is updated in the task list to recommend to end-user. The returned services are treated as the associated services to the new task in the task list. If the recommended task exists in an ad-hoc process, we update the associated services with the task. The approach for context aware service recommendation is discussed in details in [40].

3.3 Mining Task Usage Patterns

We employ Apriori algorithm [2] to identify task usage patterns from historical usage data in two steps: (1) detect the most frequently used sets of tasks (i.e., task sets); and (2) recognize task usage patterns from the frequent task sets.

Detect frequent task sets. We analyze the execution of tasks recorded in the execution instance and enumerate possible combinations of tasks of varying sizes to form candidate task sets. We start with generating initial candidate task sets of size 1, then size 2 and so forth. A task that appears in at least one execution instance becomes a candidate task set of size 1. Given the candidate task sets of size n, we generate larger candidate task sets of size n+1 in the following steps:

1) Group the candidate task sets of size n, such that all task sets in each group have n-1 tasks in common and each task set in the group has one different task. Suppose we have 5 candidate task sets of size 2 (i.e., n=2), namely, {A, B}, {A, C}, {A, D}, {B, C} and {B, D}. We divide them into 2 groups: groupA [{A, B}, {A, C}, {A, D}] and groupB [{B, C}, {B, D}]. All task sets of size 2 in each group have 1 (i.e., n-1) task in common.

2) Expand the candidate task sets in each group obtained in step 1) from size n to size n+1. As aforementioned, the task sets of size n in a group contain n-1 common tasks and 1 different task. We maintain the n-1 common tasks in each task set. To expand the task sets to size n+1, we generate the pair-wise combinations among the different tasks and merge a combination with the common tasks to form a task set. For example, groupA contains three task sets: {A, B}, {A, C} and {A, D}. The task sets have one common task A. Each task set in groupA contains a different task (i.e., B, C and D). The pair-wise combinations among the different tasks are {B, C}, {B, D} and {C, D}. The task sets of size 3 are generated by merging {A} with each combination. The resulting task sets of size 3 is {A, B, C}, {A, B, D} and {A, C, D}.

3) Calculate the frequency of a newly generated candidate task set appearing in all execution instances. Specifically, a support is defined as the number of execution instances that contain the candidate task set. For example, if a task set, {A, B, C} occurs in 16 execution instances, its support is 16. We filter out infrequent candidate task sets using a support threshold.

4) Repeatedly generate larger candidate task sets until no larger candidate task sets can be achieved. We rank all candidate task sets by their supports and select the ones with the highest support as a frequent task set.

Generate task usage patterns. We recognize task usage patterns as rules which can be represented as a context and an effect. The context and effect specify different sets of tasks correspondingly. When a context is satisfied, the effect will also take place. For example, if the task set {"car rental", "check map"} is executed, it is likely that the task set {"check weather"} will be executed. A task usage pattern describes the associations between a context and an effect. We represent task usage patterns in the format of "context→effect" (e.g., context task set {$task_2$, $task_3$… $task_N$}→ effect task set {$task_1$}). In the previous example, we represent the pattern as "{car rental, check map} →{check weather}". We recommend new tasks by matching the existing task list with the context task set of a pattern. If an existing task list matches the context task set of a pattern, we recommend the tasks in the effect task set of the pattern to extend the task list.

The strength of the task usage pattern is measured by the following equation.

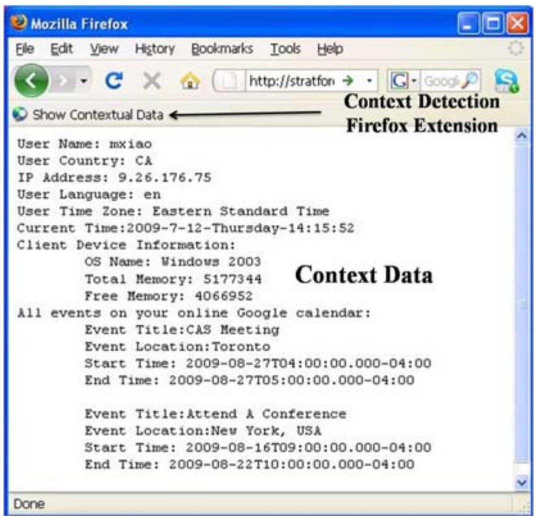
$$strength = \frac{support\ of\ (context\ taskset \cup effect\ taskset)}{support\ of\ context\ taskset}$$

Essentially, the equation evaluates the frequency that the context task set and the effect task set are executed together. The strength is in the range of 0 to 1. The higher the strength is, the more frequent the task usage pattern is exhibited. For example, strength of 50% indicates that when the context task set is executed, there is 50% probability that the effect task set will be executed. Using a threshold value, we can ensure that only the strong pattern is used to recommend tasks to end-users.

4 Implementation

To evaluate the feasibility of our proposed approach, we built a prototype to help end-users generate ad-hoc processes. The prototype is developed in Java. We use IBM Mashup Center [18] as the service Mashup platform. IBM WebSphere Service Registry and Repository (WSRR) [20] are adopted by our prototype to register and manage services. In our current implementation of the prototype, the ontologies are manually searched using Swoogle [36] and Freebase [15], and imported into our ontology database. We implemented a generic ontology description model as the internal ontology representation. The generic ontology description model captures the information needed by our framework regardless the ontology languages. An ontology parser is implemented using OWL API [29] to read the ontologies described by OWL and RDF then convert the ontologies into the generic ontology description model. The current version of our prototype has four features: context detection, task list generation, new task recommendation and task execution order detection.

***Context detection*:** To demonstrate the functionality of our context-aware service recommendation, we design and develop a Firefox extension to capture the contextual data when an end-user uses the browser. The annotated screenshot is shown in Fig. 6. The Firefox extension can detect the contextual data from the Windows operating


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://stratfon
Show Contextual Data
Context Detection Firefox Extension
Context Data
Done

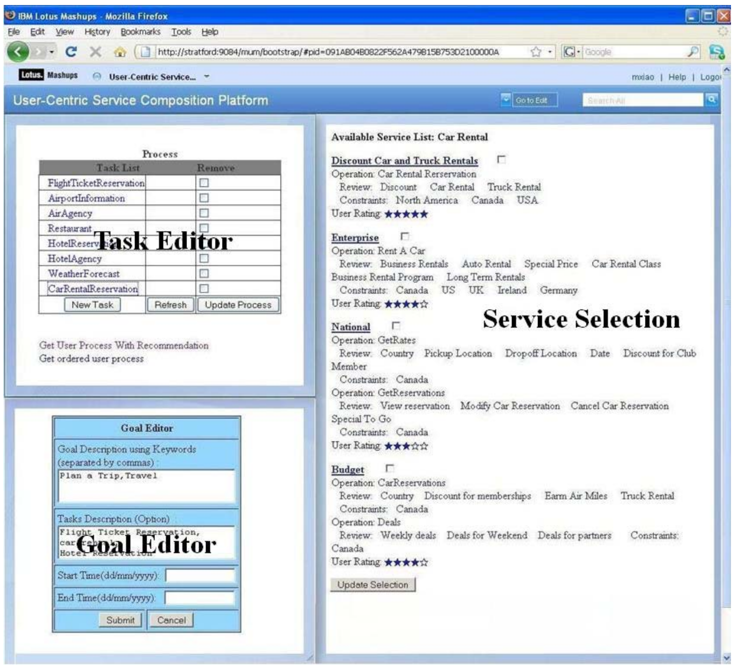
```
User Name: mxiao
User Country: CA
IP Address: 9.26.176.75
User Language: en
User Time Zone: Eastern Standard Time
Current Time:2009-7-12-Thursday-14:15:52
Client Device Information:
        OS Name: Windows 2003
        Total Memory: 5177344
        Free Memory: 4066952
All events on your online Google calendar:
        Event Title:CAS Meeting
        Event Location:Toronto
        Start Time: 2009-08-27T04:00:00.000-04:00
        End Time: 2009-08-27T05:00:00.000-04:00

        Event Title:Attend A Conference
        Event Location:New York, USA
        Start Time: 2009-08-16T09:00:00.000-04:00
        End Time: 2009-08-22T10:00:00.000-04:00
```

Fig. 6. Annotated Screenshot for Context Detection

system; track the visited websites, bookmarks and Google online Calendar. Our context-aware recommendation approach can dynamically process different context types and values. Therefore, we can easily add new context sensors to detect contextual data from the applications and devices other than the browser.

***Task list generation*:** To automatically generate a task list, we use the goal description (i.e., keywords) to find the matching ontology and decompose the goal into a set of tasks as described in section 3.1. Fig. 7 is an annotated screenshot for composition UI. An end-user can specify their goal (e.g., plan a trip to New York) in the *Goal Editor*. A task list relevant to the goal is automatically generated and presented in the *Task Editor*. A task in the task list can be associated with one or more services. As shown in Fig. 7, once an end-user selects the "*Car Rental*" task in the *Task Editor*, the associated services are automatically displayed in the *Composition UI* on the right side of the markup page. We allow an end-user to select and invoke the desired services. An end-user can edit the task list in the *Task Editor*. For example, an end-user can remove a task if it is not needed by selecting the "*Remove*" check box. An end-user can also add a new task by specifying keywords for searching for services. We record the modifications as the end-user's preferences. When an end-user specifies the same goal, our prototype provides the previously refined task list.

New service recommendation: Due to the diversified requirements of different end-users, the tasks generated from ontologies may not contain all the desired services. We need to recommend new tasks to refine the task list. In our prototype, we use two different ways to recommend new services: (1) we mine the historical usage data to identify task usage patterns to recommend new tasks and the associated


IBM Lotus Mashups - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://stratford:9084/mum/bootstrap/#pid=091AB04B0822F562A479B15B753D2100000A
Google
Lotus. Mashups
User-Centric Service...
mxiao | Help | Logo
Go to Edit
Search All
Task Editor
Goal Editor
Service Selection

User-Centric Service Composition Platform

Process

Task List	Remove
FlightTicketReservation	☐
AirportInformation	☐
AirAgency	☐
Restaurant	☐
HotelReserv[illegible]	
HotelAgency	☐
WeatherForecast	☐
CarRentalReservation	☐

New Task | Refresh | Update Process

Get User Process With Recommendation
Get ordered user process

Goal Editor
Goal Description using Keywords (separated by commas):
Plan a Trip,Travel
Tasks Description (Option):
Flight Ticket Reservation,
car [illegible]
Hotel Reservation
Start Time(dd/mm/yyyy):
End Time(dd/mm/yyyy):
Submit | Cancel

Available Service List: Car Rental

Discount Car and Truck Rentals ☐
Operation: Car Rental Rerservation
Review: Discount Car Rental Truck Rental
Constraints: North America Canada USA
User Rating ★★★★★

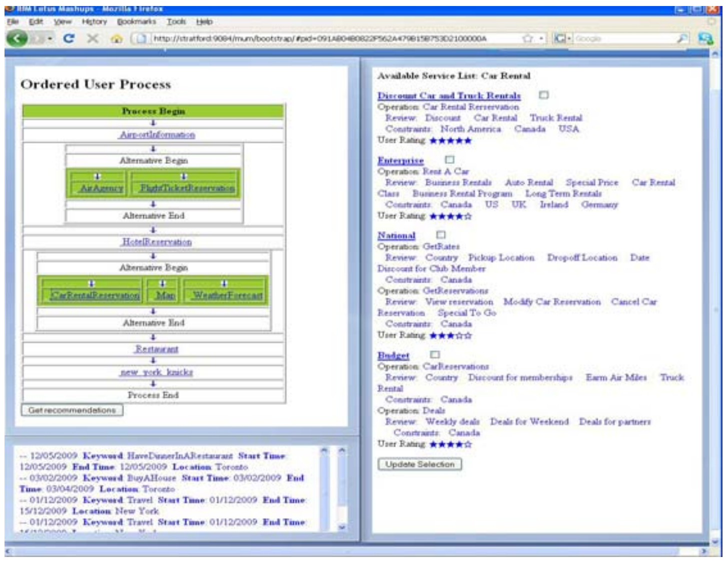
Enterprise ☐
Operation: Rent A Car
Review: Business Rentals Auto Rental Special Price Car Rental Class Business Rental Program Long Term Rentals
Constraints: Canada US UK Ireland Germany
User Rating ★★★★☆

National ☐
Operation: GetRates
Review: Country Pickup Location Dropoff Location Date Discount for Club Member
Constraints: Canada
Operation: GetReservations
Review: View reservation Modify Car Reservation Cancel Car Reservation Special To Go
Constraints: Canada
User Rating ★★★☆☆

Budget ☐
Operation: CarReservations
Review: Country Discount for memberships Earn Air Miles Truck Rental
Constraints: Canada
Operation: Deals
Review: Weekly deals Deals for Weekend Deals for partners Constraints: Canada
User Rating ★★★★☆

Update Selection

Fig. 7. Annotated Screenshot for Service Composer Panel



Available Service List: Car Rental

Discount Car and Truck Rentals ☐
Operation: Car Rental Rerservation
Review: Discount Car Rental Truck Rental
Constraints: North America Canada USA
User Rating: ★★★★★

Enterprise ☐
Operation: Rent A Car
Review: Business Rentals Auto Rental Special Price Car Rental Class Business Rental Program Long Term Rentals
Constraints: Canada US UK Ireland Germany
User Rating: ★★★★☆

National ☐
Operation: GetRates
Review: Country Pickup Location Dropoff Location Date Discount for Club Member
Constraints: Canada
Operation: GetReservations
Review: View reservation Modify Car Reservation Cancel Car Reservation Special To Go
Constraints: Canada
User Rating: ★★★☆☆

Budget ☐
Operation: CarReservations
Review: Country Discount for memberships Earn Air Miles Truck Rental
Constraints: Canada
Operation: Deals
Review: Weekly deals Deals for Weekend Deals for partners
Constraints: Canada
User Rating: ★★★★☆

-- 12/05/2009 **Keyword**: HaveDinnerInARestaurant **Start Time**: 12/05/2009 **End Time**: 12/05/2009 **Location**: Toronto
-- 03/02/2009 **Keyword**: BuyAHouse **Start Time**: 03/02/2009 **End Time**: 03/04/2009 **Location**: Toronto
-- 01/12/2009 **Keyword**: Travel **Start Time**: 01/12/2009 **End Time**: 15/12/2009 **Location**: New York
-- 01/12/2009 **Keyword**: Travel **Start Time**: 01/12/2009 **End Time**:

Fig. 8. Suggested execution order among tasks by analyzing historical usage data

services as described in Section 3.3; (2) we analyze an end-user's context to recommend services with the response to an end-user's context.

Task execution order detection: To guide end-users to navigate through the task list to fulfill the goal, we detect the task execution order from historical usage data and generate the control flow among tasks. Fig. 8 is a screenshot of the suggested execution order of tasks in the ad-hoc process.

5 Related Work

Automatic service composition. In current research, two major approaches provide automatic service composition: model driven service composition and goal driven service composition. In the model driven approach, the design document, such as UML (Unified Modeling Language) diagrams, workflows, and abstract business process, are transformed to executable business processes. For example, Pistore et al. [30][31] propose an approach to automatically generate an executable BPEL process from abstract BPEL process. In the goal driven approach, a goal for the service composition is expressed using either formal or informal languages. Process models are created on the fly to realize the goal. Most approaches in this category treat service composition as an Artificial Intelligence (AI) planning problem [21][32]. 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 of AI planning is the current state of the world; the target state is an end-user's goal; and the actions are available services. The approach proposed by Wu et al. [39] is an example of the goal driven approach. Wu et al. use an AI planning called SHOP2 to automatically compose DAML-S Web services. SHOP2 decomposes the goal into smaller and smaller sub-goals, until a Web service is found to fulfill the sub-goal. However, most of existing approaches require formal semantic description of abstract business processes or services, such as the pre and post conditions for each service. Therefore these approaches have limited support for dynamic services composition. Especially, WSDL services do not have formal semantic descriptions. Our approach reflects the changing context of an end-user without the pre-defined processes or tasks. In the goal driven approach, the goal could be described based on the input and output of services. For example, Hu et al. [17] describe the goal as the desired input and output of the composite service. The process of service composition is to find a set of services which can convert the input into the desired output. To handle large-scale distributed service composition, Hu et al. use the publish/subscribe model to describe the input and output of services. In the publish/subscribe model, publishers produce data in the form of advertisements, and subscribers register their interests by issuing subscriptions. Hu et al. map service inputs to subscriptions, and service outputs to advertisements. Using the existing distributed publish/subscribe network, they can compose services in a distributed environment. Yan et al. [43] and Li et al. [22] extend the publish/subscribe model in service composition to support dynamic service discovery (e.g., discovery of Web services whose attributes can change) and execute the business processes in distributed environments. However, simply matching the interfaces of Web services cannot guarantee the correctness of functionality for the composite process. A Web services with the same input and output may have totally different functionality. Our approach aims to compose services based on the functionality of services. We enable end-users to select services while there is more than one service having the same functionality. Similar to our approach, ontologies are used in service discovery and service composition [3][4]. In [3], the services are classified using ontology to guide end-users to search for services. The approach in [4] uses ontology to semi-automatically compose services by matching the interface of individual services. Different from the aforementioned approaches, our approach identifies the required tasks from ontologies.

Supporting End -user in Service Composition. Mashups supports a manually but user friendly way to compose services without following the formal definition of business processes. Carlson et al. [9] 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. Liu et al. [24][25][26] propose a Mashup architecture which extends the SOA model with Mashups to facilitate service composition. Similar to the work of Carlson et al. [9], Liu et al. match the input with output to help end-users compose services. They also use tags and Quality of Service (QoS) to select services. Our work enhances service Mashups by providing guidance to end-users as they create their Mashups through the automatic composition of services. The aforementioned approaches gradually compose services using the matches between the input and output of service interfaces. End-users cannot preview the composition results. Our approach allows end-users to specify goals and generate the overview of the processes before services are executed.

In addition to Service Mashups, there are some other approaches which provide support for end-users to compose services. The project UbiCompForAll (Ubiquitous service composition for all users) [14][37] provides support for non-IT professionals to compose services. UbiCompForAll did an initial experiment on evaluating a service composition tool for creating mobile tourist services by 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 concrete results on providing support for end-users have been revealed by UbiCompForAll. Obrenovic et al. [28] provide a spreadsheet-based tool to help end-user compose services. A spreadsheet (e.g., Microsoft Excel) is software application that uses rectangular tables to display information. In a spreadsheet, the content is put in cells of the table, and the relations among 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 is built up on Mashup pages. Mashups are very similar to regular Web pages and are easy for non-expert end-users to manually compose services.

Context-aware service discovery and composition. Context-aware techniques have been generally used to discover and recommend services. Yang et al. [10][44] design an event-driven rule based system to recommend services according to people's context changes. Balke and Wagner [6] propose an algorithm to select Web services based on user's preferences. The algorithm starts with a general query. If there are too many results, it expands the service query with constraints using user's preferences. By adding constraints step by step, the algorithm narrows the number of returned services to a small value. To select and recommend services, those approaches need to predefine the reaction to a specific context using rules. In our work, we automatically extend the semantics of the context value using ontologies, and use the semantics to recommend services.

Mining historical usage data. Liang et al. [23] mine service association rules from service transactional data. Agrawal et al. [1] propose a technique for mining workflows from execution logs. Cook and Wolf [12] develop an approach to recover business processes from event logs. Schimm [33] recovers hierarchically structured business processes. The aforementioned approaches focus on recovering a pre-defined workflow or business process. In our work, we discover the execution order of a list of task while there is no pre-defined workflow.

6 Conclusions and Future Work

In this paper, we present a framework that dynamically generates an ad-hoc process and personalizes the process to fit it with the context of end-users. Our approach hides the complexity of SOA standards from end-users and helps an end-user fulfill their daily activities. By discovering the semantic relations among context values using ontologies, our approach can identify an end-user's needs hidden in the context values and recommend the desired services. To refine the generated ad-hoc process, we identify task usage patterns by mining the historical usage data to recommend new tasks. We also recover the execution orders among the tasks in the ad-hoc process by analyzing historical task execution data. A prototype is developed as a proof of concept to demonstrate that our approach enables end-users to discover and compose services easily.

In our current approach, end-users need to manually provide the input data to invoke services. In the future, we plan to develop an approach to automatically generate service input data by analyzing user's context, historical execution data and the output of other services. In addition, we found that the quality of generated ad-hoc processes highly depends on the quality of the ontology relevant to the goal. However, there is no existing approach to help us select the appropriate ontologies for ad-hoc process generation. In the future, we plan to improve our approach to search for relevant ontologies. To address issue of low quality ontologies, we plan to investigate the possibility of using information from other resources (e.g., online Web pages) to refine the generated ad-hoc process.

Acknowledgements

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. R. Agrawal, D. Gunopulos, and F. Leymann, "Mining Process Models from Workflow Logs," Sixth International Conference on Extending Database Technology, 1998, pages: 469–483
2. R. Agrawal, T. Imielinski, and A. N. Swami, "Mining Association Rules between Sets of Services in Large Databases," 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., United States, May 26-28, 1993, pages: 207-216
3. K. Arabshian, C. Dickmann and H. Schulzrinne, "Ontology-Based Service Discovery Front-End Interface for GloServ," Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications, Heraklion, Crete, Greece, May 31 - June 4, 2009, Pages: 684 - 696
4. I. B. Arpinar, B. Aleman-Meza, R. Zhang, A. Maduko, "Ontology-Driven Web Services Composition Platform," IEEE International Conference on E-Commerce Technology, San Diego, California, July 6-9, 2004, pages: 146-152
5. M. Baldauf, S. Dustdar and F. Rosenberg, "A Survey on Context-aware Systems," International Journal of Ad Hoc and Ubiquitous Computing, Volume 2, Issue 4, June 2007, pages: 263-277

6. W. T. Balke, M. Wagner, “Towards Personalized Selection of Web Services,” In Proceedings of the International World Wide Web Conference (WWW 03) 2003, Budapest, Hungary, 2003, pages: 725-733
7. D. Beckett, B. McBride, “RDF/XML Syntax Specification (Revised)”, W3C Recommendation (2004), available at http://www.w3.org/TR/rdf-syntax-grammar/, last accessed on March 10, 2010
8. J. Bhogal, A. Macfarlane, P. Smith, “A Review of Ontology based Query Expansion,” *Informaltion Processing and Management*, Volume 43, Issue 4 (july 2007), 2007, pages: 866-886
9. M. P. Carlson, A H. H. Ngu, R. M. Podorozhny, L. Zeng, “Automatic Mash Up of Composite Applications,” International Conference on Service Oriented Computing (ICSOC) 2008, Sydney, Australia, December 1-5, 2008, pages: 317-330
10. I. Y. L. Chen, S. J. H. Yang, J. Jiang, “Ubiquitous provision of context aware Web services,” IEEE International Conference on Services Computing (SCC) 2006, Chicago, USA, September 18-22, 2006, pages: 60-68
11. R. Chinnici, J. J. Mreau, A. Ryman, S. Weerawarana, “Web Services Description Language (WSDL) Version 2.0,” W3C Recommendation, June 26, 2007, available at: http://www.w3.org/TR/wsdl20/, last accessed on March 10, 2010.
12. J.E. Cook, and A.L. Wolf, “Event-based detection of concurrency”, Sixth International Symposium on the Foundations of Software Engineering, Lake Buena Vista, Florida, USA, November 3-5, 1998, pages: 35-45.
13. Expedia, http://www.expedia.com/, last accessed on March 10, 2010.
14. J. Floch, E. Stav, and E. Blakstad, “Compose Your Own City Guide,” VERDIKT Conference, Oslo, Norway, Novermber 3-4, 2009, available at: http://www.sintef.no/project/UbiCompForAll/UbiCompForAll%20City%20Guide%20-%20verdikt_conf_abstract.pdf, last accessed on June 17, 2010
15. Freebase, http://www.freebase.com/, last accessed on March 10, 2010.
16. Google, http://www.google.com, last accessed on March 22, 2010
17. S. Hu, V. Muthusamy, G. Li, H. Jacobsen, “Distributed Automatic Service Composition in Large-Scale Systems,”, Distributed Event-Based Systems Conference (DEBS) Rome, Italy, July 1-4, 2008, pages: 233-244
18. IBM Mashup Center, http://www-01.ibm.com/software/info/Mashup-center/, last accessed on March 10, 2010.
19. IBM WebSphere Integration Developer (WID), http://www-01.ibm.com/software/integration/wid, last access on March 15, 2010
20. IBM WebSphere Service Registry and Repository, http://www-01.ibm.com/software/integration/wsrr/, last accessed on March 10, 2010
21. U. Küster, M. Stern, B. König-Ries, “A Classification of Issues and Approaches in Service Composition,” International Workshop on Engineering Service Compositions, 2005.
22. G. Li, V. Muthusamy, and H. Jacobsen, “A Distributed Service Oriented Architecture for Business Process Execution,”, ACM Transaction on the Web, Vol. 4, No. 1, January 2010, Article 2 (33 pages)
23. Q. Liang, J. Chung, S. Miller, Y. Ouyang, "Service Pattern Discovery of Web Service Mining in Web Service Registry-Repository," IEEE International Conference on E-Business Engineering, , Shanghai, China, October 24-26, 2006, pages: 286-293
24. X. Liu, G. Huang, H. Mei, “Towards End User Service Composition,” 31st Annual International Computer Software and Applications Conference, Beijing, China, 2007, pages: 667-678
25. X. Liu, G. Huang, H. Mei, “A User-Oriented Approach to Automated Service Composition,” 2008 IEEE International Conference on Web Services (ICWS), Short paper, Beijing, China, September 23-26, 2008, pages: 773-776

26. X. Liu, Y. Hui, W. Sun, H. Liang, "Towards Service Composition Based on Mashup," IEEE Congress on Services, 2007.
27. J. Montgomery, Microsfot Popfly: Building Games without a CS Degree, available at: http://expression.microsoft.com/en-us/cc963994.aspx, last accessed on April 3, 2010.
28. Z. Obrenovic and D. Gasevic, "End-User Service Composition: Spreadsheets as a Service Composition Tool," IEEE Transactions on Service Computing, vol. 1, No. 4, October-December, 2008
29. OWL API, http://owlapi.sourceforge.net/, last accessed on March 24, 2010
30. M. Pistore, A. Marconi, P. Bertoli and P. Traverso, Automated Composition of Web Services by Planning at the Knowledge Level, International Joint Conference on Artificial Intelligence (IJCAI) 2005, Pasadena, California, USA, pages: 1252-1259
31. M. Pistore, P. Traverso, P. Bertoli, A. Marconi, Automated Synthesis of Composite BPEL4WS Web Services, International Conference on Web Services (ICWS) 2005, Orlando Florida, USA, July 11-15, 2005, pages: 293-301
32. J. Rao and X. Su, "A Survey of Automated Web Service Composition Methods," First International Workshop on Semantic Web Services and Web Process Composition, San Diego, CA, USA, July 2004, pages: 43-54
33. G. Schimm, "Generic linear business process modeling", Proceedings of the ER 2000 Workshop on Conceptual Approaches for E-Business and The World Wide Web and Conceptual Modeling, volume 1921 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2000, page: 31-39
34. M. K. Smith, C. Welty, McGuinness, D. L., "OWL Web Ontology Language Guide," W3C Recommendation (2004), available at http://www.w3.org/TR/owl-guide/, last accessed on March 10, 2010
35. T. Strang, and C. Linnhoff-Popien, "A context modeling survey," The First International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham, England, September, 2004
36. Swoogle, http://swoogle.umbc.edu/, last accessed on March 10, 2010
37. UbiCompForAll - Ubiquitous service composition for all users, http://www.sintef.no/Projectweb/UbiCompForAll/Home/, last accessed on June 17, 2010
38. Web Services Business Process Execution Language Version 2.0, available at: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, last accessed on March 10, 2010.
39. D. Wu, B. Parsia, E. Sirin, J. Hendler and D. Nau, "Automating DAML-S Web Services Composition Using SHOP2," 2nd International Semantic Web Conference (ISWC) 2003, Sardinia, Italy, 9-12 June, 2003, pages: 195-210
40. H. Xiao, Y. Zou, J. Ng, L. Nigul, "An Approach for Context-aware Service Discovery and Recommendation," Proc. The 8th International Conference on Web Services (ICWS 2010), Miami, Florida, USA, July 5-10, 2010
41. H. Xiao, Y. Zou, R. Tang, J. Ng, L. Nigul, "An Automatic Approach for Ontology-Driven Service Composition," IEEE Intl. Conference on Service-Oriented Computing and Applications 2009, Taipei, Taiwan, December, 2009, pages: 17-24
42. Yahoo! Pipes, http://pipes.yahoo.com/pipes/, last accessed on March 10, 2010.
43. W. Yan, S. Hu, V. Muthusamy, H. Jacobsen, L. Zha, "Efficient Event-based Resource Discovery," ACM Distributed Event-based Systems Conference (DEBS) 2009, Nashville, TN, USA, July 6-9, 2009
44. S. J. H. Yang, J. Zhang, I. Y. L. Chen, "A JESS-enabled context elicitation system for providing context-aware Web services," Export Systems with Applications, Volume 34, Issue 4 (May 2008), pages: 2254-2266