An Automatic Approach for Service Composition by Mining Process Knowledge from the Web

Bipin Upadhyaya[1], Ying Zou[1], Shaohua Wang[1], Joanna Ng[2]

[1]Queen's University, Kingston, Canada

{bipin.upadhyaya, ying.zou}@queensu.ca,
shaohua@cs.queensu.ca
[2]CAS Research, IBM Canada Software Laboratory, Markham, Canada

jwng@ca.ibm.com

Abstract. Current approaches in Service-Oriented Architecture (SOA) are challenging for users to get involved in the service composition due to the in-depth knowledge required for SOA standards and techniques. To shield users from the complexity of SOA standards, we automatically generate composed services for end-users using process knowledge available in the Web. Our approach uses natural language processing techniques to extract tasks. Our approach automatically identifies services required to accomplish the tasks. We represent the extracted tasks in a task model to find the services and then generate a user interface (UI) for a user to perform the tasks. Our case study shows that our approach can extract the tasks from how-to instructions Web pages with high precision (*i.e.,* 90%). The generated task model helps to discover services and compose the found services to perform a task. Our case study shows that our approach can reach more than 90% accuracy in service composition by identifying accurate data flow relation between services.

Keywords: task model, service composition, Web, instructions, UI generation

1 Introduction

Software is prevalent in all aspects of our lives, such as checking a stock price, finding a doctor and buying a product. Nowadays a significant part of any software system is structured using software services and implemented using Web service technologies. Programmable Web[1] alone has indexed more than 9000 services that are used in our daily activities, such as education, dating, shopping, and job search. However, a single service cannot fulfill a user's goal. For example if a user wants to plan a trip, he may require a flight booking and a hotel reservation services. One or more services are combined to fulfill his goal. A service composition is a process that combines a set of logically related services to achieve a given goal. In the current state of practices, the combination process is based on a pre-defined process model that describes the services to accomplish a task, such as planning a travel. A significant amount of effort from industry and academics focuses on providing infrastructures,

adfa, p. 1, 2013.

1. ProgrammableWeb - Mashups, APIs, and the Web as Platform, http://www.programmableweb.com/

languages and tools to compose the services. However, the complexity of state-of-the-art Web services technology prevents users with limited IT skills from getting easy access to Web services and their offered functionalities. A user has to perform a sequence of tasks to complete a process. From the implementation point of view, a task can be implemented by one or more services. For example, a typical process of buying a movie ticket includes tasks related to "searching for movies", "choosing the date and time", and "paying for the ticket". Web service composition [11] requires the resolution of multiple dependencies between input parameters (IP), output parameters (OP) and non-trivial reasoning about the composition of required functionalities from smaller units of Web services. It is challenging to acquire the complete knowledge of a domain (*e.g.*, hotel booking, flight booking in travel) and then searching, and combining the services found. We envision two challenges for a novice designer or a user to perform service composition (SC) as listed below:

- **Lack of complete knowledge about the tasks involved in order to accomplish a goal.** A user has to repeatedly search the Web to learn and complete different tasks required to achieve a goal. For example, searching for a movie, finding the showtime in a local theater and making online payment are tasks for buying a movie tickets. This process is tedious, error-prone and time consuming. Knowledge from business processes to describe the tasks for achieving a goal, if available, is hard for a novice designer or a user to understand.
- **Difficult to identify a set of services and link the services to execute a process.** There are a large number of services available on the Web. Locating a suitable set of services and linking the identified services are challenging even for experienced developers. Current work in service flow identification [1, 2 and 6] does not help to identify tasks as those methods are solely based on input and output parameters of services. A user communicates with a task through a user interface (UI). Current approaches in UI generation for Web services [10 and 12] are based on technical descriptions and therefore, are difficult to understand and error prone.

In this research, we address the aforementioned challenges. The Web contains a lot of well-written instructions (such as eHow [27] and Wikihow [28]) to teach people how to perform a process (such as how to buy a camera, how to make a restaurant reservation and how to buy a movie tickets). These instructions often describe how to accomplish a sequence of tasks step by step. Our approach understands the human-written instructions and guides a user to complete a process by discovering and integrating different services. Our goal is to build a knowledge base for a process using text mining techniques that exploit the structure of the how-to descriptions. Our approach automatically identifies a task model from written instructions in the Web pages. We use a task model to identify and combine services to execute a process. To complete a process, a user needs a UI. Building a UI for a task is time consuming. We present an approach to generate a UI to execute a task. Our UI generation approach considers data-sharing between services, so a user only needs to provide minimum input for a task.

The remainder of this paper is organized as follows. Section 2 gives an overview of our approach and provides in detail discussion about each step. Section 3 presents

our case study. Section 4 discusses the related work. Finally, Section 5 concludes the paper and explores the future work.

2 Background

Our research is primarily focused on extracting knowledge from Web pages with how-to instructions. In this section, we will discuss the basic structure of how-to instruction Web pages, Web services, and task models.

2.1 How-to Instruction Web pages

How-to instructions in the Web are a knowledge base of human activities. These websites currently store millions of articles on how to do things step by step, which collectively cover almost every domain of our daily lives, such as business, education, and travel. More or less all how-to instruction Web pages have the similar format to present the content. Fig. 1 shows an example of a Web page with how-to instructions from an eHow website. The article describes the steps to buy a movie ticket online. Fig. 1 contains four annotated parts, such as a process; a short description of the process; other related processes; and a list of tasks for completing the process. We observed three types of how-to instruction articles in the Web. First type of how-to articles helps a user to perform labor-intensive processes, such as how to clean a TV screen, and how to create a contact group on iPhone. The second type contains particularly descriptive and well-defined processes, such as Web pages related to recipes or hobbies. The third type describes the dynamic processes with many choices since the parameters and choices of a task involved vary from user to user. Examples of these tasks are reserving a seat in a restaurant, and going for a trip to Europe. Our approach cannot help with the first type of labor intensive processes. For the second and the third type of how-to instructions, our approach can assist in finding possible services and integrate services to perform a process.

Fig. 1. An annotated eHow article[1]

2.2 Web Services

A Web service is a software module designed to help interoperation between machines over the Web. There are currently two approaches for interfacing to the Web with Web services, namely Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) [20]. REST as the architectural style of the Web improves scalability of the Web applications using statelessness, performance based on caching, and compatibility through content types. RESTful services [21] simplify the development, deployment and invocation of Web services. In this paper, we refer to a

1 How to Buy Movie Tickets Online | Ehow, www.ehow.com/how_2106555_buy-movie-tickets-online.html

service as a RESTful resource. If a service is not RESTful, it is transformed into a RESTful service using the techniques described in our earlier work [4]. We model a service in terms of service name, HTTP verb, input parameters and output parameters.

In a service invocation chain, there can be different linkages: user-to-service; service-to-service and service-to-user. When a user invokes a service, the relation is a user-to-service linkage. A service-to-human linkage indicates that a service needs human inputs or confirmation. A service-to-service linkage occurs when a service invokes another service.

2.3 Task Model

A Task model describes the logical tasks that have to be carried out in a process to reach a user's goals. Several task modeling notations exist, such as Hierarchical Task Analysis (HTA), Goals, Operators, Methods, Selection rules (GOMS), and ConcurTaskTrees (CTT). CTT [30] is a graphical notation that supports the hierarchical structure of tasks, which can be interrelated through a powerful set of operators (such as iteration and optional) that describe the temporal relationships (such as choice, interleaving and enabling) between subtasks. CTT describes four types of tasks: the user, the application, their interaction and the abstract tasks. W3C [19] specification for CTT [30] provides complete information on CTT Meta-model and relations among the tasks. We use CTT to model tasks as CTT follows an engineering approach to task models. Moreover, the semantics of CTT are more complete and precise. It can support automated service discovery more effectively than other user task modeling formalisms. CTT is applied in the field of end-user programming. It is easy for end-users to express and understand CTT.

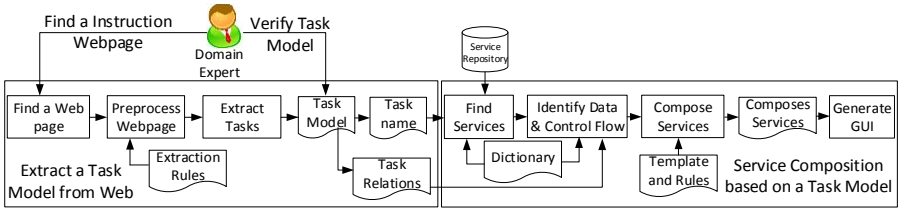

Fig. 2. Overall steps to generate UI for a task from Web services

3 Overview of Our Approach

A user selects one of many how-to instruction Web pages related to his goal. Our approach is to search and compose services based on a user selected Web page. A process is well-defined, concrete action that a user performs to achieve a goal. In order to achieve a user's goal, we break a process into a set of tasks that can be scheduled and completed. Fig. 2 shows the overall steps of our approach. We identify task models from how-to instructions in the Web. We use the task models to find and compose services. We describe each step in details in this section.

3.1 Extracting a Task Model from the Web

In this sub-section, we introduce our approach that automatically extracts a task model from a semi structured Web page.

Find Web Pages.

In this paper, we examine two specific Web sites to extract human written instructions (eHow [28] and Wikihow [27]). If a Web page matches the user's scenario, we use this Web page to extract a task model to form service composition.

Preprocess How-to Instruction Web Pages.

To analyze Web pages, we parse Web pages to build a Document Object Model (DOM) tree structure. An HTML file may contain mismatched HTML tags although it can be correctly displayed by Web browsers due to the fault-tolerance capability of Web browsers. We use HTML syntax checker [9] to correct the malformed HTML tags. Then we parse the HTML into DOM tree structure. The preprocessor contains two steps: 1) learn rules from the sample Web pages; and 2) apply rules on a Web page to extract tasks. We manually examine and learn the DOM structure of the title and the instruction steps. We use these learned DOM structures to form the rules. This procedure is based on the assumption that the documents collected from an identical source share the common structures. The preprocessor uses wrapper induction approaches to extract the title and the instruction steps from a Web page. The title becomes the root of a task model and the instruction steps become the tasks to complete a process (*i.e.*, extracting tasks).

```
Input: InstructionSteps
Output: Action Object Lists
Algorithm
1. For each step in InstructionSteps
2. step=Instructionsteps[index]
3. List<Segment> seglist=ComputerSegment(step);
4.      For all Seglist sleseglists do
5.              ASegment=POSTagger(sl);
6.              List<ActionObject> aolist=ExtractActionObject(ASegment)
7.              List conjection=ExtractConjuction(Asegment)
8.              Aobj=Order(aolist, conj)
9.      End For
10. End For
```

Fig. 3. An algorithm for identifying tasks from how-to instruction Web pages.

Extract Tasks.

Each instruction step describes the tasks in a process. We extract functional semantics of an instruction step. Functional semantics express the main intent of a sentence in terms of the action-object pairs. An action-object pairing articulates what action is performed with an object. An action is represented by a verb and objects are described by nouns in a sentence. For example, in a sentence "buy a ticket", the functional semantic pair is {buy, ticket}. "buy" and "ticket" correspond to the action and the object respectively. Our objective is to extract tasks involved in a process (*i.e.*, instruction steps) as action-object pairs. We analyze the instruction steps to extract tasks as functi

-onal semantics. Fig. 3 shows the algorithm to identify tasks. An instruction steps may contain multiple segments. Each segment is an instruction to perform a task. A segment is a sentence in the instruction step containing one or more verbs and nouns. Line 3 in Fig. 3 extracts the segments from an instruction step. We use a well-stabilized part-of-speech (POS) tagger [29] to identify the syntactic structure of a segment. Step 2 in Fig. 4 shows a POS tagged sentence. The number (*i.e.*, inside small brackets) in Step 2 indicates the index of the word as a noun or a verb in a sentence. For example in Step 2 of in Fig. 4, selection is the first noun (NN) and the movie is the second noun (NN). We post-process the generated data structures to resolve object names consisting of multiple words (*e.g.*, "Computer screen"), phrasal verbs (*e.g.*, "go to"), and pronominal referrals (*e.g.*, "it"). We assume "it" always refers to the last mentioned object, which is proved to be a sensible heuristic in most of the cases. For each segment, we extract actionable verb (VB)/verb phrases (VP) and the related noun phrases (NP). (VB/VP) or (NP) forms a task. Step 3 shows action-object pairs, such as "confirm selection", "reviewing movie", "reviewing theater", "reviewing showtime" and "reviewing computer screen".

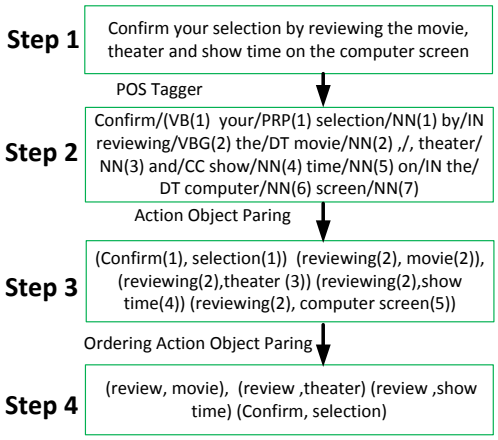

Fig. 4. An example for task extraction steps

$$PMI(domain, task) = \frac{hits(\text{domain } AND\ task)}{hits(task)} \quad (1)$$

where hits (x) is the number of results that a search engine returns for a query 'x'.

We filter irrelevant tasks from the list of tasks extracted. PMI (Pointwise mutual information) [23] helps identify how two words or phrases are related based on all the information available in the Web. We use Google Services[1] to calculate PMI. The PMI score is the number of hits for a query that combines domain and task divided by the hits for the task alone. This can be viewed as the probability that a domain and a task can be found on a Web page as shown in equation 1. For example in Step 4, we compute PMI for each term with the domain "movie" and each object in action-object pairs. As a result, PMI (movie, review theater) is 0.287; PMI (movie, confirm selection) is 0.197; and PMI (movie, review computer screen) is 2.13415E-05. To identify the threshold to filter irrelevant tasks, we randomly selected 10 how-to description Web pages and manually check the result PMI. We empirically set 10^{-3} as the threshold. In line number 6 of Fig. 3, we compute the score. If the score is above the threshold, we consider it as an action-object pair. Hence in Fig. 4, (review, computer screen) is filtered out. We convert the verb to the base form. For the extracted pairs, we analyze its occurrence index and compare with clauses, such as before, after, and by, to identify the order of the sequence. We further arranged the remaining action-

1 https://developers.google.com/web-search/docs/

object (review movie, review theater and review selection) pairs based on the hierarchical relationship among the words. If no relation is found, the arrangement is based on the order of their occurrences in a sentence.

Using the action-object extracted from Fig. 3, we build a CTT task model. Fig. 5 shows a task model to buy a movie ticket online. Only two kinds of temporal relationship exist in how-to instructions. If the two noun phrases are connected through and clauses, there is a sequential enabling info (*i.e.*, []>>) relation. If the noun phrases are connected with or clauses there is a Choice (*i.e.* []). If the same task appears multiple times in a process, we choose the last place where the task had appeared and remove other occurrences. If a step contains multiple tasks, we make an abstract task which connects all the tasks of a step. An extracted model is verified by a user. It is easy to add and modify tasks in a CTT model.

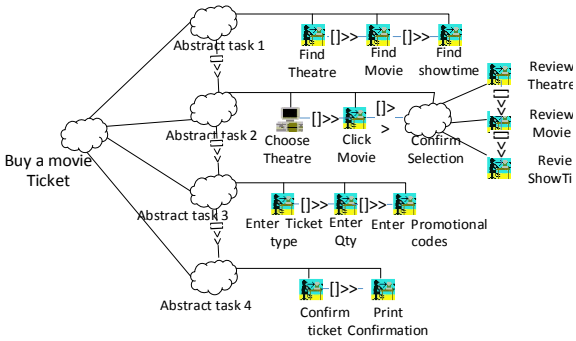

Fig. 5. A simplified task model extracted from Fig. 1

Table 1. Different information extracted from the task model

Field	Heuristic and Explanation	Example
Domain Concept	Noun in the process title	In "How to Buy Movie Tickets Online", movie is the domain of process title.
Service Name	For each task, the first word corresponds a service name.	In "Review movies", review is related to the service name
Input	Noun occurring with the UI related words (Input, Enter, Fill, Click, Submit,)	In "Enter Name", name is the word related to input
Output	Noun occurring with the UI related words (show, select, read, confirm, validate, check, review, decide, ensure, choose)	In "Show movie list", movie list is the word related to the output.

3.2 Service Composition based on the Task Model

In this sub-section, we present our approach to find and compose services based on the task model generated from Section 3.1.

Find Services.

In this step, we find the services for each step in CTT task model. We use our previous work in concept-based service discovery [3] to discover services. A concept is a semantic notion or a keyword for describing a subject, *e.g.*, "traveling" or "taxi reservation". Service repository indexes services based on concepts available in the service

description documents. Our approach clusters services using the concepts conveyed in the service description. While searching for a service, we perform the entity identification [15] and change the query if the services are not found. For example if our service repository does not contain a service for “Buy a camera” then we change the query to a more general upper level domain “Buy a product”. We use WordNet [5] as the knowledge base for transformation. WordNet [5] is a lexical database that groups words into a set of synonyms and connects words to each other via semantic relations. For the task model shown in Fig. 4, we identify concepts to search for services. Table 1 lists the concepts extracted from the task model.

Table 2. Rules used to decompose words.

Rule	Before applying rule	After applying rule
CaseChange	FindCity	Find, city
	getMovie	Get, Movie
Suffix containing No.	City1	City
Underscore separator	Customer_Information	Customer, Information
Dash separator	Find-city	Find, City

Identify Control and Data Flows.

We consider the task relations from the CTT model as a control flow of a composite service. We find the data-dependency between services and change the control flow based on the data-dependency. A data-dependency graph depicts the collaboration relations between services related to different tasks. Each service has a name and takes input parameters and gives an output. Either input or output of a service can be empty, but, not both. Multiple input and output messages in a composite service are merged into a set of input or output. We exclude fault messages as they seldom contribute to the data flow to the subsequent services. Similarly, access keys for services are excluded as they do not contribute to the data flow.

$$Semantic(p_{s1}, p_{s2}) = \begin{cases} 1 \; if \; p_{s1} \; and \; p_{s2} \; are \; \text{identical/synonymous} \\ \frac{1}{\#links} \; if \; p_{s1}, p_{s2} \; have \; hierarchical \; relation \\ 0 \; otherwise \end{cases} \quad (2)$$

where, p_{s1} and p_{s2} are the name of parameters of services s1 and s2;
#link is the number of nodes to reach a common parent from the names of p_{s1} and p_{s2} in WordNet

The name of a service and the input/output parameters follow the conventions used in programming languages. Table 2 shows the rules to decompose input parameters and output parameters. After decomposing words, we use porter stemmer [20], which is the process for reducing derived words to their stem, base, or root form. For example, the words "fishing", "fished", "fish", and "fisher" have the same root word, "fish". Equation (2) shows a formula to calculate the semantic similarity. For each word, we calculate the semantic similarity. WordNet helps to identify if two terms are semantically similar and to what degree they are similar. When the words are identical or synonymous, the semantic similarity is 1. If there is a hierarchical relation, it depends

on their similarity degree. If the semantic (p_{s1}, p_{s2}) is greater than a threshold (*i.e.*, 0.3), we consider it is a match. We choose a threshold by analyzing different service input and output parameters. For each pair of services, we evaluate the semantic similarity between the input and output parameters. Based on the similarity between parameters, we identify the linkage between different services.

For a set of services, if an output parameter of one service and an input parameter of another service are semantically similar, a data flow relation exists. We compute the semantic weight between the services which is the sum of semantic similarity between all parameters of two services. The semantic weight is normalized in interval 0 to 1. 0 means no data flow. 1 indicates that all the inputs of a service come from the output of another service. Fig. 6 shows a simplified version of the dependency graph containing services of different tasks. Moreover, the direction of the edges between two enclosed nodes dictates the dataflow. The service where the arrow points to takes one output of the preceding service as its input. For example shown in Fig. 6, service B2 of task B has two links between services, C1 and C2 on task C.

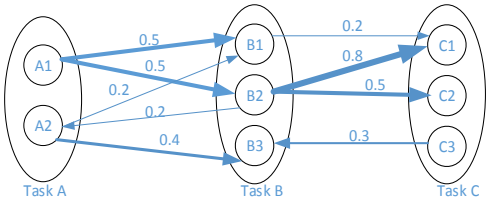

Fig. 6. Dependency graph between different services in three different tasks

Compose Services.

We have a task model that defines the steps that a user needs to follow to perform a process. A task model helps us to find relevant services and then gives a logical flow between different services. However, a task is performed by services. The execution order of services can be different from the task model. It depends on the data dependency between services. An executable task is a graph G (V, E) where G is a Directed Acyclic Graph (DAG). Each vertex $v_i \in V$ is a service. Each edge (u, v) represents a logical flow of messages from service u to service v. If there is an edge (u, v), then it means that an output message produced by service u is used to create an input message to service v. Our goal in this step is to combine one or more services in the dependency graph to form a task that can maximize the data sharing properties between services. From Fig. 6, we select A1 between {A1, A2} based on the semantic weight. We select B2 among {B1, B2} because its weight to C2 or C1 is the maximum. Hence the flow will be A1→B2→C1.

The task model from how-to instruction Web pages, such as recipe Webpage is different. Obviously services for the action-object pairs like "Boil Pasta" are not available, and hence there is no flow information. For such Web pages, we define a service composition template and use a rule to invoke predefined template in such a specific domain. We impose a rule to invoke E-commerce templates for recipe related how-to Web pages.

Generate User Interface (UI).

A user needs an interface to provide the data to perform a task. Each task may require more than one service. As mentioned in the background section, three relations

among services are defined. We want to increase the service-to-service interaction to minimize the demand for a user to enter information for accomplishing a task. Our service selection approach maximizes data sharing by choosing services with the maximum shared parameters. We extract all the parameters that may be required by a service that is to be executed later and ask a user to fill the information avoiding multiple services-to-the user or user-to-service linkage.

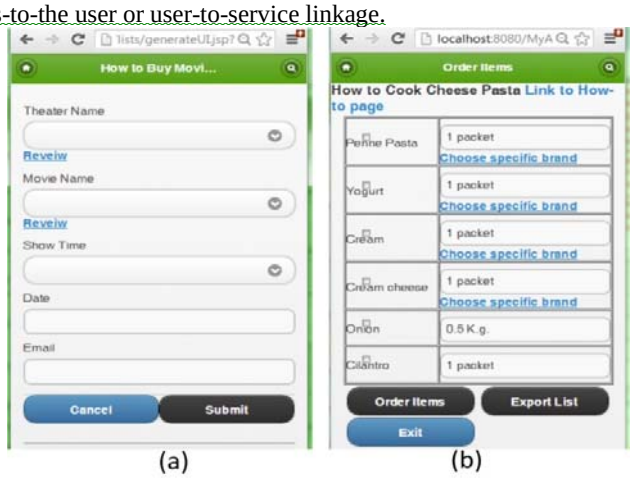

Fig. 7. Screenshot of generated UI

We enhance Kasarda *et al.* [22] approach to generate UI when each service has different input and output parameters. The approach by Kasarda *et al.* is simple and easy to implement. The use of XHTML makes the generated UI adaptable to cross-platforms. Decision for the placement input or output in UI is based on the dependency between the services. The service input and output parameters are represented in XML. We enhance the approach by Kasarda *et al.* to find the relation among different input elements. Our enhancement also helps decide whether the input element should be user editable or not. We enhance the UI generation techniques using the following techniques:

1. If the output of a service is not used as an input parameter to another service, the UI element of the output parameter is not user editable;
2. If the output of a service is a single parameter and used as an input parameter to another service, the UI element of the parameter is not user editable;
3. If the output of a service has multiple values (*i.e.*, array) and one of the elements is used as an input parameter to another service, the UI element of the parameter is user editable;
4. If the input parameter for a service does not come as an output from any other services, we select an appropriate UI element based on the approach described in [22].
5. If a node has multiple paths and if they do not merge to the same node later, the shortest path becomes a new task and used as a link. For example in Fig.7a review movie and theater are tasks used as links.
6. Based on the dependency found in section *Identify Control and Data Flow* step, we link service invocation with UI elements. Unless it's predefined in a template, we

add two UI elements (submit and cancel), submit executes all the service invocations and the cancel exits the service composition. For example as illustrated in Fig. 7, the change in theater name alters the values in the selection box for the movie name which in turn modifies the values in the show time selection box.

UI elements for an executable task are dependent on each other. A change in one parameter can trigger a run-time change in another UI element. We also identify the dependency between these parameters. Fig. 7 shows the generated UI. Fig.7a shows a UI of a process to get a movie ticket. Fig. 7b illustrates a UI to order items to prepare a recipe based on pre-defined e-commerce template.

4 Case Study

We conduct a case study to evaluate the effectiveness of our approach. The objectives of our case study include: 1) evaluate if our approach can achieve high precision and recall to extract task models from how-to instruction Web pages; and 2) evaluate the accuracy of our approach to compose services from task models.

4.1 Setup

We collect 40 different how-to instructions from eHow and Wikihow. The collected Web pages were from different domains, such as communication (*e.g.*, send SMS), finance (*e.g.*, find a stock price of a company), and E-commerce (*e.g.*, buy a product). We avoid selecting many processes from the domain to ensure the case study result is not skewed due to this particular. In addition, we collected more than 600 service description files to examine our approach on the service composition capability. The collected service description files have more than 4,000 different services. Our case study specifically answers the following two research questions.

1. How effective is our approach to extract a task model from Web pages?
2. How accurate is our approach of service composition based on task models?

In our case study, the first author evaluated all the result. Our evaluator has two years of experience in developing Web services and composite services.

4.2 Evaluate the effectiveness of our approach to extract task models.

We measure the effectiveness of our approach on identifying task models using precision and recall. As shown in equation (3), the precision is the ratio of the total number of tasks correctly extracted by our approach to the total number of tasks in a how-to instruction Web page. Recall, as shown in equation (4), is the ratio of the total number of tasks correctly extracted by our approach to the total number of tasks existing in the how-to description Web pages.

$$precision = \frac{\{\text{relevant tasks}\} \cap \{retrieved\ tasks\}}{\{\text{retrieved tasks}\}} \quad (3)$$

$$recall = \frac{\{\text{relevant tasks}\} \cap \{retrieved\ tasks\}}{\{\text{relevant tasks}\}} \quad (4)$$

Table 3 presents the effectiveness of our approach to extract task models from how-to instruction Web pages. Table 3 shows the effectiveness of task identification of our approach. Our approach has the average precision of 90% and average recall of 59%. The reason for lower recall is due to verb scoping during task extraction step. When two verbs are conjoined, it is not clear whether the noun is associated with both or just the latter one. With "review and buy a camera" and "Go and buy a camera", for example, our approach needs to decide whether the noun (*i.e.*, camera) is associated with either both the verbs or just the one. Our approach could not correctly identify multi-word expressions (MWEs) in the instruction steps. MWEs represent the structure and meaning that cannot be derived from the component words, as they occur independently. Examples of MWEs include conjunctions like 'as well as' (meaning 'including'), and phrasal verbs like 'find out' (meaning 'search').

Table 3. Results for our case study

Domain	Effectiveness of our approach to extract task model			Accuracy of our approach to compose services	
	#Web pages	Precision (%)	Recall (%)	# task Model	Accuracy of SC (%)
Hotel	10	91	57	3	88
Flight	10	89	53	2	85
Ecommerce	8	92	61	4	83
Finance	7	88	58	3	85
Communication	5	93	70	3	95

4.3 Evaluate the accuracy of automatic SC based on the task model.

We are interested in evaluating the accuracy of service composition based on the task model. Equation (5) gives the measure of our accuracy. Accuracy of our approach is given by the ratio of the correctly identified data and control flow by the total number of data and control flows among services.

$$Accuracy = \frac{\#\text{Correctly Identified } flow\ for\ SC}{\#Flow \text{ required } for\ SC} \times 100\ (\%)\ (5)$$

To check the accuracy of service composition we first made sure that there are services available to perform the service composition for the task models. We selected the task model with at least one candidate service to form the service composition. For this case study, we selected 15 out of 40 task models. We manually verified the services used by our approach and the data and control flows between the services selected. Table 3 presents the result of our case study for automatic service composition based on a task model. Our approach has average accuracy of 85% to identify correct flow different services. The uses of ambiguous words and the words not available in WordNet cause difficulties in identifying semantic similarity. Some element names were misspelled or inconsistently named. *e.g.*, an element *conferenceIdentifier* was misspelled as *conderenceIdentifier* in several places in a conference management service. Hence our approach could not identify the data flow between services. Some

of the entities are named differently, *e.g., ASIN* and *OfferListingID* were used interchangeably in Amazon Product API. We were unable identify these interchangeable entities.

4.4 Threats to Validity

In this section, we discuss the limitations of our approaches and the threats that may affect the validity of the results of our case study. In our case study, only one of the co-author inspects the results. Our evaluator has experience in developing service composition and has knowledge of the domains used in our case study. The manual verification introduces bias because a single evaluator could make mistakes. We should have recruited additional people for the evaluation. Unfortunately, we were not able to recruit more evaluators with sufficient knowledge about service-oriented applications and who can spend considerable time to manually inspect our results. To generalize our results to task extraction and service composition in other domains, we chose to study systems with a variety of domains to help ensure the generality of our results. Even though we think the case study in still need to enhance to include more domains and a wider variety of tasks from each domain.

5 Related Work

Our work is related to three research areas: mining human activities from the Web, Web service composition and UI generation from Web services.

5.1 Mining Human Activities from the Web

Singh *et al.* [21] collect knowledge about commonsense including daily living is the Open Mind Common Sense project. More than 729,000 raw sentences representing commonsense knowledge collected from the general public through a template-based Web interface. In our previous work [16], we extract process knowledge by analyzing the menus and forms that are limited to a certain domain. In this work, we extract human activity knowledge automatically from how-to instructions on the Web. We define a process in terms of a sequence of tasks. Our approach is similar to Perkowitz *et al*. [13] who proposed a method for mining human activity models in terms of a sequence of objects involved in an activity and their probabilities. From the definitions of activities obtained in external resources, such as how-to instructions, recipes, and training manuals, they attempted to extract objects by identifying noun phrases and their hyponyms under 'object' or 'substance' categories in WordNet. Unlike their approach, we not only focus on identifying tasks to achieve a goal. We find the corresponding services and link services to execute a task.

5.2 Web Service Composition

For composing services, a user can perform either a manual composition in cooperation with domain experts or automatic composition [2, 6 and 7] conducted by software programs. In the manual approach, human users who know the domain well

(*e.g.*, domain ontology) select proper Web services and weave them into a cohesive workflow. Although users may rely on some GUI-based software [24] to facilitate the composition, in essence, it is a labor-intensive and error-prone task. On the contrary, in the automatic composition approach, software programs know if two Web services can be connected or not (*i.e.*, via syntactic matching of Web services parameters or even via semantic matching). The problem with AI based approach is how we can make sure what are the services need to perform a task, such as "planning a holiday in New York". Our approach combines both manual and automatic aspects of SC. We use task models to discover relevant services and define the sequence between services. Instead of relying on data dependencies among service, our approach uses task relations along with data-dependencies to identify service sequences.

Currently available end-users SC [11] tools, such as Yahoo! Pipe [35] and IBM Mashup center [25] provide a user friendly environment for end users to integrate different services. However, those products require end-users to manually identify all the services to form a process. Our approach reduces the workload of end users by automatically generating processes required based on easily available knowledge in the Web. Our approach helps novice programmers and end-users.

5.3 User Interface Generation from Web Services

The creation of user interfaces is still a complex problem. Bias *et al.* [8] state that 50% of time for building an application is due to the UI development. There are some approaches for generating the UI automatically [6, 10 and 12]. These approaches focus on the generation of user interfaces for services. A common way to generate user interfaces for services is their inference from services description, like WSDL [17] or WADL [18] files. Given that data types can be matched to specific graphical controls, the inference mechanism to create UI forms can be straightforward. However, the inference mechanism is limited to a certain degree because the developer may need to include more details to the controls on the form that cannot be inferred from technical descriptions. Some research, such as Dynvoker [12] cannot support composed services. Moreover, a designer cannot edit the resulting forms. We present an approach for task identification that considers the composition as a dynamic hyperlinked environment of services. Our UI is editable making modification easier.

6 Conclusion and Future Work

We provide an approach to build task models from how-to instruction Web pages. Our case study shows that our approach has high precision to identify instructions from Web pages. In most cases, our approach can correctly identify tasks. Similarly, we use the task model to find relevant services to execute tasks based on the data flows between the services. Given a correct task model, our approach can build an executable process with 90% accuracy. We believe the UI generation process is still a complex issue. The manual creation process is time consuming and complex because it requires the combination of the work from application developers and the UI designers. We designed and developed a tool which intends to ease and enhance the

automatic UI generation process. In the future, we plan to conduct a user study to see the effectiveness of UI generation. We would also like to perform a larger case study including different how-to instructions.

References

[1] C.E. Gerede, R. Hull, O.H. Ibarra, and J. Su: Automated composition of e-services: lookaheads. ICSOC, 2004, pages 252- 262

[2] J. P. Thomas, M. Thomas, and G.Ghinea. "Modeling of web services flow." E-Commerce, 2003. CEC 2003. IEEE International Conference on E-commerce, 2003.

[3] B. Upadhyaya, F. Khomh, Y. Zou, A. Lau, J. Ng, A concept analysis approach for guiding users in service discovery. In SOCA, 2012 5th IEEE International Conference on (pp. 1-8).

[4] B. Upadhyaya, Y. Zou, H. Xiao, J. Ng and A. Lau. Migration of SOAP-based services to RESTful services, In Proc. of the 13th IEEE International Symposium on WSE, pp. 105-114, 2011

[5] G. A. Miller. WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11 (pp). 39-41.

[6] L. Li, and Wu Chou. "Automatic Message Flow Analyses for Web Services Based on WSDL." IEEE International Conference on Web Services, 2007.

[7] S.Y. Hwang, E.P. Lim, C.H. Lee; C.H. Chen, "Dynamic Web Service Selection for Reliable Web Service Composition", IEEE Transactions on Services Computing, vol.1, no.2, pp.104,116, 2008

[8] R. G. Bias, and D.J. Mayhew, Cost-Justifying usability. San Francisco: Morgan Kaufmann Publishers

[9] JTidy, http://jtidy.sourceforge.net/

[10] M. Kassoff, D. Kato, W. Mohsin, Creating GUIs for web services. IEEE Internet Comp. 7 (5), 66–73.

[11] N. Mehandjiev, A. Namoune, U. Wajid, L. Macaulay, A. Sutcliffe, "End User Service Composition: Perceptions and Requirements," IEEE 8th European Conference on ECOWS, pp.139,146, 2010

[12] J. Spillner, M. Feldmann, I. Braun, T. Springer, A. Schill, A., Ad Hoc Usage of Web Services with Dynvoker, Towards a Service-Based Internet. LNCS 5377, Springer, pp. 208–219, 2008

[13] M. Perkowitz, M. Philipose, K. P. Fishkin, D. J. Patterson: Mining models of human activities from the web. WWW 2004: 573-582

[14] B. Raphael, G. Bhatnagar, I.F. Smith,Creation of flexible graphical user interfaces through model composition. Artif. Intell. Eng. Des. Anal. Manuf. 16 (June (3)), 2002,173–184

[15] T. Poibeau and L. Kosseim, Proper Name Extraction from Non-Journalistic Texts. Proc. Computational Linguistics in the Netherlands., 2001.

[16] H. Xiao, B. Upadhyaya, F. Khomh, Y. Zou, J. Ng and A. Lau. An automatic approach for extracting process knowledge from the Web. In Proc. of ICWS. pp. 315-322, 2011

[17] Web Service Definition Language (WSDL), www.w3.org/TR/wsdl

[18] Web Application Description Language, www.w3.org/Submission/wadl/

[19] World Wide Web Consortium (W3C), http://www.w3.org/

[20] R. T. Fielding, R. N. Taylor, Principled design of the modern Web architecture. pp. 407–416.

[21] L. Richardson, S. Ruby, RESTful web services, 2007

[22] K. Ján, M. Necaský, and T. Bartoš, Generating XForms from an XML Schema., NDT (2) 2010: 706-714

[23] P.D. Turney, Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. Proceedings of the Twelfth European Conference on Machine Learning. Springer-Verlag, Berlin, 491-502

[24] IBM WebSphere Integration Developer, http://www-01.ibm.com/software/integration/wid/

[25] IBM Mashup Center, https://greenhouse.lotus.com/wpsgh/wcm/connect/lotus+greenhouse/lotus+greenhouse+next+site/home/products/ibm+mashup+center

[26] iGoogle, http://www.google.com/ig

[27] wikihow-how to do anything, http://www.wikihow.com/Main-Page

[28] eHow | Discover the expert in you., http://www.eHow.com

[29] D. Klein, C.D Manning, Accurate Unlexicalized Parsing. In: 41st Annual Meeting of the Association for Computational Linguistics, pp. 423–430 (2003)

[30] Concur Task Trees (CTT), http://www.w3.org/2012/02/ctt/.

[31] P. Singh, T. Lin, E. T. Mueller, G. Lim, T. Perkins and W. L. Zhu,Open Mind Common Sense: Knowledge acquisition from the general public, First International Conference on Ontologies, CA.

All URLs are last accesses on 25-July-2013