Mining User Intents to Compose Services for End-Users

Yu Zhao, Shaohua Wang, Ying Zou
Queen's University, Kingston, Ontario, Canada
yu.zhao@queensu.ca, shaohua@cs.queensu.ca,
ying.zou@queensu.ca

Joanna Ng, Tinny Ng
IBM CAS Research, Markham, Canada
{jwng, tng}@ca.ibm.com

***Abstract*—End-users repetitively perform various on-line tasks and invoke multiple web services for their re-occurring activities, such as planning a trip. Usually, end-users have to complete different tasks in order to achieve a goal, and look through large volumes of services to find the best ones that satisfy their constraints, such as a budget limit. Current approaches on service composition require programming skills and domain knowledge to accomplish goals. Moreover, existing approaches lack an automatic way to analyze end-users' goals and extract relevant tasks for achieving goals. In this paper, we provide a lightweight service composition framework for end-users with limited technical background. Our framework analyzes end-users' goals expressed in natural languages to mine tasks (*e.g.*, plan a trip) and non-functional constraints (*e.g.*, budget $<$ 500). Our framework extracts task models from textual descriptions of tasks (*e.g.*, eHow, a How-to instruction website) to guide the selection of services and recommend web services that can finish tasks and satisfy constraints. We have designed and developed a prototype as a proof of concept. We conduct case studies to evaluate the effectiveness of our framework. Our framework can identify tasks with a precision of 93% and a recall of 77%, and extract non-functional constraints with a precision of 89% and a recall of 76%. A user study shows that our framework is helpful for end-users to compose services.**

***Index Terms*—end-user goals and constraints, natural language processing, web tasks, service composition**

I. Introduction

Web services are prevalent in our daily life. ProgrammableWeb[1], a service repository, indexes more than 14,000 web services. Such a large volume of web services enable end-users to conduct various on-line tasks to achieve end-users' goals, such as shopping on-line and planning a trip. Intuitively, an end-user's goal can be expressed in natural languages. For example, one friend can send a message "I want to plan a Disney World trip with a budget less than \$500" to another friend. This message, a textual description, conveys an end-user's goal that has a task and a constraint. A task represents the main intent of the goal, *e.g.*, plan a Disney World trip, and constraints show end-users' non-functional requirements, *e.g.*, "price $<$ 500". Generally speaking, a single web service is not sufficient to accomplish an end-user's goal. End-users have to decompose an end-user's goal into a set of relevant tasks at a lower granularity to discover the relevant services that can carry out the corresponding tasks, and compose a set of logically related web services to accomplish the end-user's goal.

[1]http://www.programmableweb.com/

An extensive research effort has focused on assisting end-users in service composition. Approaches such as [5][7][16][26] rely on programming logics to orchestrate services and data sources. Such approaches require in-depth knowledge and technical skills. Mashup technology provides a user-friendly infrastructure for end-users to combine different services [14][17][28]. However, end-users have to specify relevant tasks to achieve a goal and repetitively search for services that can achieve the relevant tasks. Moreover, the aforementioned approaches do not consider end-users' constraints in service composition. End-users experience the following pain points to perform on-line tasks:

- **Unable to automatically analyze end-users' goals and identify relevant tasks for given goals.** End-users need some knowledge of relevant tasks and a plan to achieve the tasks for a given goal. For example, to plan a trip to Orlando Disney, end-users need know the relevant tasks, such as buying admission tickets, booking hotels and purchasing flight tickets. In the current practice, end-users issue queries manually and search the web for relevant web services. To relieve end-users from cognitive overloading, we need to provide end-users techniques for automatically analyzing end-users' goals and obtaining relevant tasks.
- **Repetitively searching for desired web services.** To find the best service among functionally similar services, end-users have to look through a large amount of services. For example, an end-user can visit various ticket sale websites to find the greatest deal for Disney tickets. Moreover, the end-user has to be cautious about the constraints on selecting services. For example, in the above example, the end-user needs to limit all of the expenses within \$500. It is a tedious and time-consuming job to repetitively search for services to meet the constraints. Therefore, there is an urgent need for automatically discovering and composing related web services based on the constraints.

To guide end-users to accomplish their goals, our earlier work by Upadhyaya *et al.* [25] proposes an approach to extract the relevant tasks from on-line How-to instructions. However, the approach lacks the analysis of end-users' goals (*e.g.*, in natural languages) for mining tasks and constraints. Moreover, the approach does not take into consideration end-users' constraints in service selection and integration.

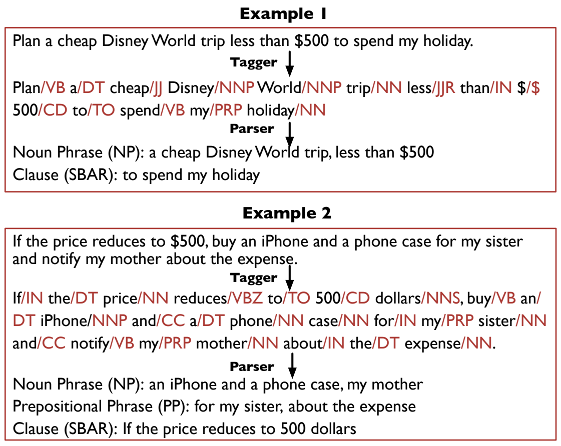
In this work, we extend the earlier work [25]. Our framework allows end-users to express their goals in natural languages without relying on any programming language and technical terms. Our framework understands user intents using natural language processing techniques. We leverage textual descriptions of tasks, such as on-line How-to instruction web pages, to automatically discover a collection of relevant tasks for guiding the selection of web services. Compared with the approach of task identification in [25], our approach is more sophisticated by analyzing grammatical structures of sentences. Moreover, we recommend sets of services that match end-users' constraints.

We have implemented a prototype of our framework as a proof of concept. We evaluate the effectiveness of our framework through empirical experiments and a user case study. On average, we achieve a precision of 93% and a recall of 77% on identifying tasks, and a precision of 89% and a recall of 76% on identifying constraints from natural languages. The results of our case study show that our framework is helpful for end-users in service composition.

Paper Organization: Section II presents the background of this paper. Section III describes our proposed framework that automatically analyzes user intents and recommends services. Section IV introduces our empirical studies. Section V summarizes the related studies. Section VI concludes our work and describes the future work.

II. Background

In this paper, we analyze textual descriptions of tasks, such as on-line How-to instruction web pages to generate relevant tasks using natural language processing (NLP) techniques. Here, we discuss the NLP tools used by our framework and basic structures of How-to instruction web pages.


Example 1

Plan a cheap Disney World trip less than $500 to spend my holiday.

Tagger

Plan/VB a/DT cheap/JJ Disney/NNP World/NNP trip/NN less/JJR than/IN $/$ 500/CD to/TO spend/VB my/PRP holiday/NN

Parser

Noun Phrase (NP): a cheap Disney World trip, less than $500
Clause (SBAR): to spend my holiday

Example 2

If the price reduces to $500, buy an iPhone and a phone case for my sister and notify my mother about the expense.

Tagger

If/IN the/DT price/NN reduces/VBZ to/TO 500/CD dollars/NNS, buy/VB an/DT iPhone/NNP and/CC a/DT phone/NN case/NN for/IN my/PRP sister/NN and/CC notify/VB my/PRP mother/NN about/IN the/DT expense/NN.

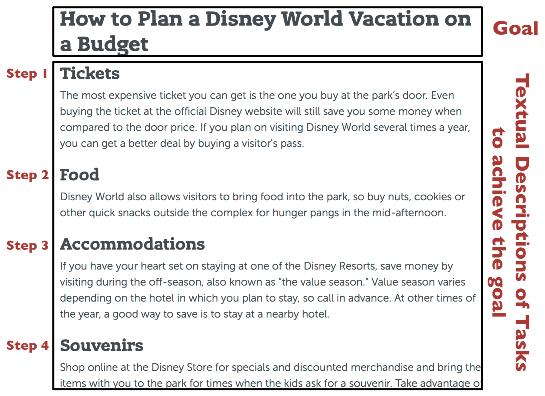
Parser

Noun Phrase (NP): an iPhone and a phone case, my mother
Prepositional Phrase (PP): for my sister, about the expense
Clause (SBAR): If the price reduces to 500 dollars

Fig. 1: Two examples of POS tagger and Parser analyzed sentences (POS tags: determiner (DT), verb (VB), noun (NN), preposition (IN), to (TO), conjunction (CC), adjective (JJ), comparative adjective (JJR), adverb (RB), number (CD) and personal pronoun (PRP))

1) Natural Language Processing: We analyze the grammatical structure of a sentence using the state-of-the-art Stanford Part-of-Speech (POS) Tagger [24] and Stanford Natural Language Parser [9]. Figure 1 shows two examples of using the POS tagger and the Parser for analyzing sentences. POS tagger tags each word in a sentence with part-of-speech markers. For example, in Figure 1, "plan" is tagged as a verb and "cheap" is tagged as an adjective. The Parser groups words into phrases with brackets. For example, "a cheap Disney World trip" is grouped as a noun phrase (*i.e.*, NP) in the Example 1 of Figure 1. Another noun phrase "less than $500" expresses an end-user's requirement about the goal. The Example 2 shows a more complex sentence that contains three tasks: "buy iPhone", "buy phone case" and "notify mother expense".

2) On-line How-to Instructions: How-to instruction websites, such as eHow[2] and wikiHow[3], are knowledge bases that teach people to conduct activities in various aspects of human lives, such as education, travel and finance. Typically, a How-to instruction web page describes a list of steps that need to be executed for accomplishing a goal. Figure 2 shows an example of a How-to instruction web page containing steps for planing a Disney World trip[4]. The How-to instruction web page contains a goal and textual descriptions of tasks to achieve the goal.



How to Plan a Disney World Vacation on a Budget

Tickets

The most expensive ticket you can get is the one you buy at the park's door. Even buying the ticket at the official Disney website will still save you some money when compared to the door price. If you plan on visiting Disney World several times a year, you can get a better deal by buying a visitor's pass.

Food

Disney World also allows visitors to bring food into the park, so buy nuts, cookies or other quick snacks outside the complex for hunger pangs in the mid-afternoon.

Accommodations

If you have your heart set on staying at one of the Disney Resorts, save money by visiting during the off-season, also known as "the value season." Value season varies depending on the hotel in which you plan to stay, so call in advance. At other times of the year, a good way to save is to stay at a nearby hotel.

Souvenirs

Shop online at the Disney Store for specials and discounted merchandise and bring the items with you to the park for times when the kids ask for a souvenir. Take advantage o

Fig. 2: An annotated screen shot of an eHow How-to instruction web page

III. Overview of Our framework

Figure 3 shows the steps of our framework. An end-user enters a goal description into our framework, as shown in Figure 4. We analyze the goal description to extract tasks and constraints. If an end-user's goal description is high-level and coarse-grained, we expand the goal to extract a collection of relevant tasks from textual descriptions of tasks. We discover proper services from our service pool based on the extracted relevant tasks. We incorporate the constraints extracted from the goal description (*e.g.*, price $<$ 500) into the selection of services and recommend the selected services to end-users, as shown in Figure 5.

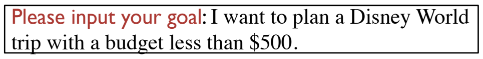

Please input your goal: I want to plan a Disney World trip with a budget less than $500.

Fig. 4: Goal input box in our framework

[2]http://www.wikihow.com
[3]http://www.ehow.com
[4]www.ehow.com/how_4716518_plan-disney-world-vacation-budget.html

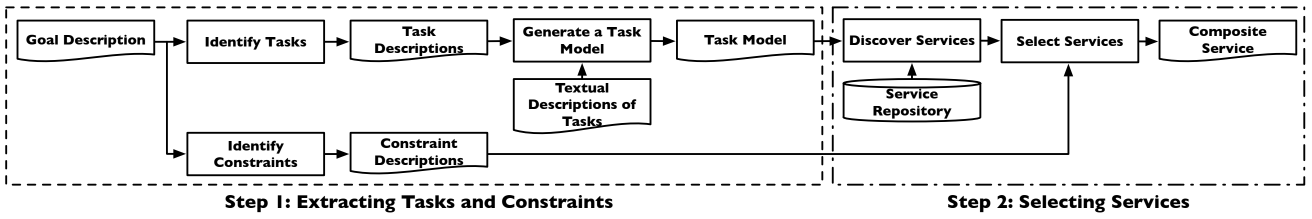


Fig. 3: An overview of our framework

TABLE I: Rules to extract tasks in various linguistic patterns (VB stands for Verb, NP denotes Noun Phrase and PP represents Prepositional Phrase)

#	Linguistic Pattern	Rules to Extract Task	Example
1	VB+NP	Action: verb (VB); Object: nouns in a noun phrase (NP)	In "buy an iPhone", the task is "buy iPhone".
2	VB+PP	Action: verb (VB); Object: nouns in a prepositional phrase (PP)	In "invest in the company", the task is "invest company".
3	VB+NP+PP	Action: verb (VB); Object: nouns in a noun phrase (NP) and nouns in a prepositional phrase (PP) that has a preposition "of" or "about"	In "notify my mother about the expense, the task is "notify mother expense".

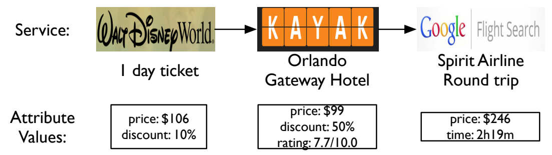

Service:
I day ticket
Orlando Gateway Hotel
Spirit Airline Round trip
Attribute Values:
price: $106 discount: 10%
price: $99 discount: 50% rating: 7.7/10.0
price: $246 time: 2h19m

Fig. 5: Output of our framework: a set of services

A. Extracting Tasks and Constraints

We analyze sentences to extract tasks and constraints. We normalize words in sentences to their stem, base or root form using the Porter stemmer [19] (*e.g.*, "reduced", "reducing" and "reduces" are normalized to "reduce").

*1) **Identifying Tasks**:* A sentence has at least a verb and a noun to express tasks and constraints. Verbs and nouns from noun phrases or prepositional phrases are formed into action-object pairs [11]. An action-object pair shows how an action can be performed on an object. For example, "plan a Disney World trip" has an action "plan" and an object "Disney World trip". We identify tasks from action-object pairs.

A sentence can have multiple tasks depending on the number of action-object pairs. For example, "buy iPhone", "buy phone case" and "notify mother expense" are three tasks for the Example 2 in Figure 1. We separate a sentence into multiple segments, and identify tasks in each segment. A valid segment should have a verb and an object. We identify segments in the following two cases:

- *Segments separated by conjunctions:* As shown in the Example 2 of Figure 1, the sentence segment before the conjunction (*i.e.*, and) contains a verb "buy" and a noun phrase "an iPhone and a phone case". "Notify my mother about the expense" following the conjunction (*i.e.*, and) is another segment.
- *Segments in a prepositional phrase or a subordinate clause:* A prepositional phrase (*e.g.*, after buying an iPhone) or a subordinate clause (*e.g.*, to spend my holiday) can have sentence segments. We identify action-object pairs and extract sentence segments.

Segments, declaring facts about various situations, are irrelevant to service composition, *e.g.*, "The hotel is in Toronto". We filter out such sentence segments based on phrasal verbs and subject noun phrases:

- *Phrasal verb:* If only one verb exists and the verb is one of {be, is, am, are, was, were, 's, 're}, we filter out the segment since the verb is not an action.
- *Subject noun phrase:* If a subject noun phrase contains a determiner, such as "a", "the" and "this", we consider that the segment does not include a task, since the segment declares facts about the referenced subject.

Table I shows our rules for extracting tasks using various linguistic patterns from segments. In pattern #1, we skip a noun phrase without a noun and identify objects from the next noun phrase. For example, "buy her an iPhone" contains two noun phrases (*i.e.*, "her" and "an iPhone"), and has a task "buy iPhone". A segment with a pattern #1, 2 or 3 can contain multiple tasks, since objects in a phrase (*i.e.*, a noun phrase or a prepositional phrase) can be separated by a comma or a conjunction. For instance, "buy an iPhone and a phone case" contains two tasks "buy iPhone" and "buy phone case". In an extracted task description (using rules from Table I), an object "this" or "it" is replaced with the last mentioned object [25].

We identify tasks from segments in a subordinate clause. Tasks in a subordinate clause, such as an infinitive-clause and a *that*-clause (*i.e.*, a clause starts with a *that*), can be used to explain the main clause. We merge tasks in the main clause and the subordinate clause in such a case. For example, Example 1 in Figure 1 has a task "plan Disney World trip spend holiday".

*2) **Identifying Constraints**:* Non-functional constraints are linked with tasks to express an end-user's desire about services [29]. Constraints can be used to 1) fill in service parameters (*e.g.*, "in Toronto" for a hotel service); 2) identify a target person (*e.g.*, for my sister); and 3) restrict on the selected services (*e.g.*, less than $500).

We identify constraints from prepositional phrases, noun phrases and clauses in a segment. Table II shows our rules for extracting constraints. We store constraints in a key-value pair form. A key defines a label for a constraint and a value shows the details of a constraint. For example, in pattern #3 of Table II, the key is an object (*i.e.*, sister) and the value contains the object and a preposition (*i.e.*, {sister, for}).

Constraints can have an upper or a lower range to restrict the selected services. When phrases contain a numerical number

TABLE II: Rules to extract constraints from Noun Phrases, Prepositional Phrases and Clauses

	#	Linguistic Pattern	Example	Key-Value pair (Key=Value)
Noun Phrase	1	Adjective	cheap	Key:Adjective Value:{Adjective}; cheap={cheap}
	2	Quantifier Phrase	less than $500	Key:Measurement Unit Value:{Number,Operator}; dollars={500, less than}
Prepositional Phrase	3	Without Number	for my sister	Key:Noun Value:{Noun,Preposition}; sister={sister,for}
	4	With Number	over 500 dollars	Key:Measurement Unit Value:{Number,Operator}; dollars={500, greater than}
	5	Preposition is "than"	price lower than iPhone	Key:Last Mentioned Object Value:{Noun, Operator}; price={iPhone, less than}
Clause	6	Verb + "to" + Number	if the price reduces to 500	Key:Measurement Unit Value:{Number, Operator}; price={500, less than}

or a preposition "than" (*e.g.*, pattern # 2, 4, 5 and 6 in Table II), we detect such constraints with the following steps:

- The key of the constraint is a measurement unit, such as $ and dollars. Symbols (*e.g.*, $) are transferred into natural languages (*e.g.*, dollars). If no measurement unit is detected, we use the last mentioned object as the key (such as pattern #6 in Table II).
- The value contains an operator that defines either a "less than" or a "greater than" relation. An operator can be inferred from adjectives (*e.g.*, less), prepositions (*e.g.*, under) and phrasal verbs (*e.g.*, reduce to). We define a bag of words that denote a "less than" relation: *less, lower, fewer, most, maximum, below, under, reduce, decrease*, and a "greater than" relation: *more, higher, least, minimum, over, increase, raise*. A word in a constraint that is synonymous with the bag-of-words determines an operator. We use WordNet [4], a large English database, to identify synonymous words. WordNet groups synonymous words into sets called *synsets* and connect synsets by different relations, such as hypernym and meronym. Hypernym denotes a "kind of" relation. For example, *bed* is a hypernym of *furniture*.

To make the extracted constraints meaningful, we recognize four types of constraints: *location* (*e.g.*, in Toronto), *time* (*e.g.*, at Christmas), *price* (*e.g.*, less than $500), and *relative* (*e.g.*, for my sister). To determine the type of a constraint, we use WordNet to identify ***hypernym trees*** for each word in the extracted ***key*** of the constraint. A hypernym tree represents a sequence of synsets, and each of synsets has a hypernym relation with the succeeding synset. The synsets of "location", "time", "monetary" and "relative" represent the conceptual semantic meaning of location, time, price and relative respectively. We traverse all the paths of the hypernym trees to record the number of occurrences of the four synsets. The synset with the highest occurrence number wins and we classify the type of the constraint as the semantic meaning of the synset. The key of the constraint that is initially identified using rules in Table II is replaced with the constraint type. For example, "dollars" is replaced with "price".

*3) **Generating a Task Model**:* To guide the service selection and execution, we automatically generate a task model to represent the logical execution order of tasks. In a task model, tasks are associated with parameters to the tasks. Parameters are nouns that relate to input or output parameters of services. For example, a task description "type room numbers" has a parameter "room numbers". We use verbs in a task description to identify parameters of tasks [25]. Verbs for input parameters are: *input, enter, fill, click, submit, type*, and for output parameters are: *show, select, read, confirm, validate, check, review, decide, ensure, choose*. An identified parameter belongs to the previous identified task.

To represent the tasks in a logical relation, we identify three kinds of task relations, *i.e.*, sequence, parallel and choice.

- **Sequence:** Tasks are performed in a sequential order. We recognize the sequential order in textual descriptions from prepositions and conjunctions, such as "before", "after", "by" and "if".
- **Parallel:** Tasks are performed in any order. "And" and "as well as" in textual descriptions denote a parallel relation.
- **Choice:** Only one task needs to be performed. We consider the conjunction "or" as a choice relation.

If the order of tasks can not be found, we arrange the tasks based on the order of occurrences in a sentence. We generate a task model using the approach described by [25]. More details of the approach can refer to [25]. An end-user can customize the generated task model to merge, modify, delete and add tasks. Figure 6 shows the generated task model for the goal of planning a Disney World trip.

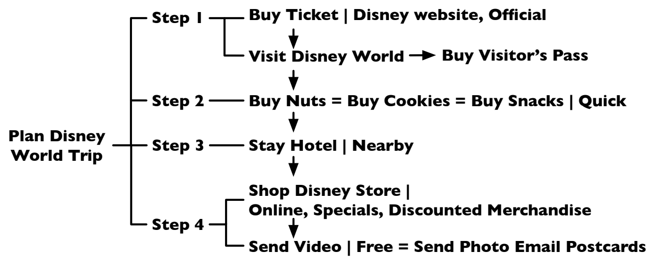

Plan Disney World Trip
Step 1
Buy Ticket | Disney website, Official
Visit Disney World
Buy Visitor's Pass
Step 2
Buy Nuts = Buy Cookies = Buy Snacks | Quick
Step 3
Stay Hotel | Nearby
Step 4
Shop Disney Store | Online, Specials, Discounted Merchandise
Send Video | Free = Send Photo Email Postcards

Fig. 6: A simplified task model extracted from Figure 2 ("|" separates a task and constraints, "$\rightarrow$": sequence order, "=": choice order).

B. Selecting Services

We select sets of services that are desired by end-users by incorporating end-users' constraints from goal descriptions. We follow the control flow specified in a task model (in Section III-A3) as the order to select services. We first discover services and then we perform the process of selecting services.

*1) **Discovering Services**:* A service in our framework has a service description, input and output parameter descriptions. We normalize words in service descriptions to identify meaningful words. We decompose words using special characters, such as "_" and "-", and capital cases if applicable. Suffix containing numbers is removed (*e.g.*, *car1* is normalized to *car*). Moreover, we use WordNet to remove non-English words. Stop words (*e.g.*, "a", "the" and "is") are removed and word stemming is performed.

We discover services using Lucene[5], a text search engine. We index services with Lucene to organize services structurally in the repository. Lucene combines Boolean Model (BM) and Vector Space Model (VSM) [15] to perform searching with queries, and derives a score for each service denoting the relevance to the query. We use a task description, parameters of tasks and a constraint description as a query. A query can contain a few keywords, such as "buy iPhone". A relevant service may not contain the keywords extracted from the query. To increase the chance of identifying relevant services, we expand keywords in queries to include semantically related concepts (words) using ConceptNet [13]. ConceptNet is a large knowledge base that connects concepts using semantic relations. ConceptNet records similarity scores between concepts to represent the validity of the relations. We choose four categories of relations, *i.e.*, part (*i.e.*, "PartOf", "DerivedFrom" and "MemberOf"), superior (*i.e.*, "HasA"), synonym (*i.e.*, "RelatedTo" and "Synonym") and instance (*i.e.*, "IsA"), to expand keywords.

- *Part* represents a "part of" relation. For example, *wing* is a part of *bird*.
- *Superior* denotes that a concept is superior to another. For example, *bird* is superior to *wing*.
- *Synonym* represents that two concepts are semantically related. For example, *iphone* is a synonym of *ipod*.
- *Instance* shows that a concept is a specific instance of another. For example, *iphone* is an instance of *telephone*.

Figure 7 shows an example of the expanded concepts for the keyword "iphone". All the retrieved concepts are included in the queries to search for services. We represent concepts as weighted vectors to denote the importance of a concept in a service. The weights for the concepts are the similarity scores between concepts and query keywords. We calculate three scores to match services with tasks using Lucene:

iphone: telephone, smartphone, camera phone, blackberry, multimedia, wireless, communication, combination, ipod, knockoff, computer, high-end, melafonino

Fig. 7: Expanded concepts for the keyword "iphone" using ConceptNet

- **Task score:** A similarity score between a task and the overall description of a service.
- **Parameter score:** A similarity score between parameters of a task and the parameter description of a service.
- **Constraint score:** A similarity score between constraints for a task and the overall description of a service.

We derive a synthetic score using Equation (1). Given a task, we retrieve top 10 services, since Silverstein *et al.* [22] find that end-users typically only go through the first 10 results.

$$SyntheticScore = S_{task} + S_{parameter} + S_{constraint} \quad (1)$$

Where S_{task} is Task Score; $S_{parameter}$ is Parameter Score and $S_{constraint}$ is Constraint Score.

2) ***Selecting Services:*** A large amount of functionally similar services can be returned for a task. For example, executing a *Kayak hotel*[6] service can return a large amount of hotels. Our framework recommends the hotel services to meet end-users' constraints. We use JDK 8.0, HTTPClient version 4.5[7] and Axis2[8] to invoke services. HTTPClient is employed to invoke RESTful services and Axis2 is used to execute SOAP-based services.

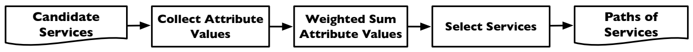


Fig. 8: An overview of our service selection process

Figure 8 shows steps of our process of selecting services. Several candidate services with the same functionality can contain different non-functional attribute values (*e.g.*, different prices and user ratings) [29]. To select a service that meets end-user's satisfaction, the non-functional attributes of services need to be considered. We collect attribute values from each service and compute an overall score using the weighted sum approach described as follows:

- **Scaling.** Some attributes can be negatively related to end-user's satisfaction. For instance, end-users prefer services with a lower price. We scale the attribute values to a range from 0 to 1 using the approach described by [29]. The higher is the value, the better is the quality.
- **Weighted Summing.** We use the following equation to derive a score:

$$Score(S_i) = \sum_{j=1}^{n} ScaledValue_{i,j} \times \omega_j \quad (2)$$

Where n is the total number of attributes; ω_j is the weight for the jth attribute. $\omega_j \in [0,1]$, $\sum_{j=1}^{n} \omega_j = 1$.

Our service selection process identifies paths of services that follow the order of tasks specified in the task model. We randomly generate weights in a service selection process. Given a number of candidate services for a task, we select the service with the maximum weighted sum score. The value of an attribute for a path of services is a sum of the attribute values of services in the path. For example, the price of planning a trip is a sum of the price of an admission ticket, a hotel and a flight ticket. If an attribute value violates an end-user specified constraint (*e.g.*, price $<$ 500), we discard the path. We perform the service selection process for multiple runs independently with different weight settings to recommend several paths to end-users.

IV. Case Study

In this section, we introduce our case study that evaluates our framework. We first present the setup of our case study. Then, we discuss the empirical study.

A. Case Study Setup

Collecting How-to instructions. We collect web pages of How-to instructions from eHow and wikiHow using Google

[5]http://lucene.apache.org

[6]www.kayak.com

[7]https://hc.apache.org/httpcomponents-client-ga/

[8]http://axis.apache.org/axis2/java/core/

Custom Search API[9]. We extract the content of instructions using the approach proposed by [25]. We collect 100 different How-to instructions from 5 domains, *i.e.*, travel (*e.g.*, plan a trip), E-commerce (*e.g.*, buy clothes), finance (*e.g.*, buy stocks), education (*e.g.*, apply a university) and job (*e.g.*, find a job). Each domain has 20 instructions.

Creating a service repository. We create a service repository with two types of services, *i.e.*, RESTful services [20] and SOAP-based services. We collect and download 1,400 public available WSDL files for SOAP-based services. We use ProgrammableWeb to manually download 401 RESTful services. The collected 1801 services fall into 5 domains: travel (595 services), E-commerce (351 services), finance (420 services), education (206 services) and job (229 services).

User study. We setup a user study to evaluate the effectiveness of our framework. We recruit 14 subjects (*i.e.*, 5 females and 9 males) to participate in our user study. All of the subjects are graduate students who do not have background on service composition techniques, but are familiar with on-line tasks. The subjects typically spend 8-10 hours on-line per day. We assign each domain with an equal number of subjects to avoid the skewed user study. For each subject, we give a face-to-face tutorial on the background knowledge and steps to conduct the user study. The tutorial lasts about 15 minutes. We ask a subject to submit a questionnaire[10] after the study. The questionnaire contains a set of single choice questions, regarding the features and usability of our framework. Each question has five choices for a subject to choose, *i.e.*, strongly disagree, disagree, neutral, agree, and strongly agree. We ask a subject to specify additional five goals in our questionnaire to be used in **RQ3**. We also ask a subject for suggestions on our framework. Each subject takes approximately 1 hour to finish the user study.

B. Research Questions

RQ 1. Can our framework effectively extract tasks and constraints from textual descriptions in natural languages?

Motivation. It is critical to accurately extract tasks and constraints specified in natural languages in order to understand end-users' goals and build task models for service selection.

Approach. We apply our framework on the collected How-to instructions to extract tasks and constraints. We evaluate our framework using precision and recall as shown in Equations (3) and (4). The precision is the ratio of the total number of correctly retrieved tasks (or constraints) to the total number of tasks (or constraints) extracted by our framework. The recall is the ratio of the total number of correctly retrieved tasks (or constraints) to the total number of tasks (or constraints) in a How-to instruction. The first author manually evaluated the results.

$$Precision = \frac{\{relevant\ items\} \cap \{retrieved\ items\}}{\{retrieved\ items\}} \quad (3)$$

$$Recall = \frac{\{relevant\ items\} \cap \{retrieved\ items\}}{\{relevant\ items\}} \quad (4)$$

TABLE III: Results for extracting tasks and constraints(P: precision;R: recall)

Domain	Tasks		Constraints	
	P(%)	R(%)	P(%)	R(%)
Travel	92	77	89	79
E-commerce	95	81	88	79
Finance	92	74	91	76
Job	93	74	88	72
Education	92	78	88	71

Results. **Our framework can effectively identify tasks and constraints from on-line How-to instructions written in English**. Table III lists the precision and recall for the identification of tasks and constraints. On average, our framework can achieve a precision of 93% and a recall of 77% for the identification of tasks, and a precision of 89% and a recall of 76% for the identification of constraints.

We summarize the main reasons for the misidentified tasks and constraints: (1) The POS tagger tools can tag a word erroneously. Therefore, we can not identify phrases correctly. For example, in "select an online road trip planner","select" should be tagged as a verb. However the tool parses "select" as a noun. (2) Some action-object pairs of tasks do not convey intents of sentences. For instance, for "make things interesting", "make things" is extracted as a task. However, it fails to carry any particular meaning. (3) A constraint can have several meanings. The variety of meanings brings us difficulties to interpret the semantic meaning of constraints. For instance, in "write a letter to the bank", "bank" represents a location. However, we tag "bank" as "price" since bank has the semantic meaning of "monetory".

RQ 2. Are end-users satisfied with our framework for guiding the service selection?

Motivation. Our framework analyzes user intents in a goal description and constructs a task model to guide end-users to select services. We are interested to evaluate end-users' perceptions of using our framework.

Approach. We conduct a user study to evaluate our framework. We have designed and implemented a prototype of our framework. To compare with our framework, we developed a baseline approach. The subjects conducted both approaches. We did not tell the subjects which is our approach.

In our framework, a subject follows the steps below:

(1) Specify goals: A subject inputs one goal for a given domain in English. A subject can modify the extracted tasks if the extracted results do not satisfy the subject's requirement.

(2) Select on-line How-to instructions: We automatically list top 10 How-to instructions. A subject chooses a relevant How-to instruction.

(3) Customize task models: We automatically generate a task model from a How-to instruction. A subject is allowed to customize the task model, *i.e.*, merging, deleting or adding tasks, if he or she wants to.

(4) Select services: we automatically retrieve services for each task from our service repository using our approach as

[9]https://developers.google.com/custom-search/
[10]http://goo.gl/forms/Y3yTq2gtmd

TABLE IV: Questionnaire results on end-users' experience about our framework(P:percentage;Number:number of subjects who agree or strongly agree)

Summary	**Questions**	**P(%)**	**Number (#/14)**
Framework Quality			
Usefulness	Is our system helpful for service composition?	93%	13
Time Saving	Does our system conserve time to compose services?	86%	12
Language Processing Quality			
Importance	Is it important to extract tasks and constraints from the goal description?	100%	14
	Is it important to automatically generate a task model?	93%	13
Accuracy	Are the tasks and constraints extracted from the goal description by our framework meaningful?	100%	14
	Is the task model extracted by our framework meaningful?	86%	12
Completeness	Do you have complete knowledge to achieve the goal using the baseline approach?	43%	6
	Does our framework remind you useful tasks?	86%	12
	Does the task model extracted by our framework cover the complete knowledge to achieve the goal?	71%	10

described in Section III-B1. A subject selects favorite services for each task.

In the baseline approach, the subject enters the same goal as using our framework. A subject needs to manually specify the needed tasks to achieve the goal. Each task is described using keywords. We discover services for each task using Lucene described in Section III-B1 same as our framework. Subjects choose their desired services for each task.

Results. **Our framework is helpful for guiding end-users in service selection.** Table IV shows the questionnaire results of our user study. Automatic task model identification relieves end-users from repeatedly composing services for the same goals. The task models extracted by our framework are meaningful for the subjects. When conducting the baseline approach, the subjects typically do not have a comprehensive knowledge to achieve a goal. Our framework reminds the subjects with useful tasks. In our framework, we identify tasks and constraints by properly analyzing end-users' goals and on-line How-to instructions in English. Overall, the subjects agree that our framework is helpful for service composition and saves subjects' time.

RQ 3. Can our framework help end-users select desired sets of services?

Motivation. When selecting multiple services with constraints, end-users need put a lot of effort to pay attention on the constraints. Our framework automatically selects end-user desired services to satisfy end-users' constraints.

Approach. We run real scenarios in two widely used domains, *i.e.*, travel and E-commerce, as a proof of concept for our framework. In each domain, we select 5 goal descriptions provided by end-users in questionnaires. We apply our framework on each goal description. We identify a task model and collect available services for each task. A task model has at least two tasks that have available services to perform service selection. We collect three kinds of service attribute values, *i.e.*, price, duration and user rating. We use the approach described in Section III-B2 to select services.

TABLE V: Subject issued goal descriptions and results of our service selection (#S: total number of services; OF: our framework; GS: gold standard; A: average accuracy)

	#	**Goal Description**	**#S**	**Time(ms)**		**A (%)**
				OF	**GS**	
Travel	1	I want to make a round-trip travel plan between Beijing and Toronto for 14 days.	396	243	1,711	82%
	2	I want to go to Vancouver for 5 days with a maximum cost of $2000.	171	195	703	67%
	3	I want to travel to India with travel cost less than 2500 dollars.	211	176	974	89%
	4	I want to plan a trip to Japan with price less than $3000.	216	175	843	77%
	5	I'm expecting a 10-day trip in USA.	239	188	799	100%
E-commerce	6	I want to buy a laptop with 16G ram.	217	107	4,307	71%
	7	I want to buy a jacket with price less than 500 bucks.	137	118	184	89%
	8	I want to buy a gift for myself.	171	123	1,541	79%
	9	I want to buy a car which is environmental friendly.	358	112	147	80%
	10	Buy a book about stock.	400	126	175	80%

To validate our recommended sets of services, we create a *gold standard* by conducting an exhaustive search of available services and generating all the possible paths of executing services. The *gold standard* contains the Pareto optimal paths that satisfy end-users' constraints. A path is Pareto optimal if-and-only-if no other paths can make one or more attribute values higher without worsening any other attributes. We compare the results from our framework with the results from the *gold standard* and compute the accuracy of our framework using Equation (5). The accuracy is the ratio of the number of the correctly identified services to the total number of services in the optimal paths of the *gold standard*. For each goal description, we run our approach 10 times to select services and compute the average execution time and accuracy.

$$Accuracy = \frac{\{\#Correctly\ Identified\ Services\}}{\{\#Services\ in\ Gold\ Standard\}} \qquad (5)$$

Results. **Our framework is more efficient to help end-users select services.** Table V shows that, on average, our framework can select 252 services using 156 milliseconds (*i.e.*, 0.156 seconds). To create a *gold standard*, the average execution time is 1,138 milliseconds (*i.e.*, 1.138 seconds). The execution time of a service selection process depends on the number of tasks and the number of services for each task [29]. Our framework has an average accuracy of 81% to identify optimal services. Our framework selects a service for a given task using the weighted sum approach. However, the weighted sum approach would fail to discover a Pareto optimal service that is located in the concave areas of the Pareto front [18]. For the scenarios #2, 3, 4, 7 listed in Table V, the identified paths for executing services can satisfy end-users' constraints.

V. Related Work

Our work is related to two research areas: natural language processing and end-user driven service composition.

Natural language processing. Natural language processing is studied and applied in various aspects, such as information extraction (IE), question-answer (QA) systems and semantic parsing. An IE system utilizes natural language processing techniques to extract predefined types of information from unstructured text [1][3][12][21]. QA systems analyze end-user questions in natural languages and search for direct answers from knowledge bases [8][10]. Thomason *et al.* [23] and Chen *et al.* [2] utilize semantic parsing techniques to translate natural languages to a formal representation form understood by machines. In our work, we analyze textual descriptions of tasks in the context of service composition. Our framework is similar to Upadhyaya *et al.* [25] who mine process knowledge to compose services. Unlike their approach, we allow users to express goals in natural languages to improve the expressiveness of goal descriptions.

End-user driven service composition. Wang *et al.* [26] propose a spreadsheet-like programming model to allow end-users build applications in Mashup. Maraikar *et al.* [16] use a coordination language to compose heterogenous services and data sources. Hang *et al.* [5] design a meta-model to assist end-users in service composition. All of the aforementioned approaches demand basic programming concepts from end-users. Approaches, such as Hornung *et al.* [6] and Hua *et al.* [27], use keywords to search for services. However, they do not identify user intents and extract constraints from end-user issued queries. Zeng *et al.* [29] take into account the constraints, expressed in an expression language, in the service composition. Different from Zeng *et al.*'s work, we extract constraints from natural languages.

VI. Conclusion And Future Work

To accomplish end-users' goals, end-users need take a hassle to discover relevant web services and compose them based on end-users' constraints. To shield the complexity of service composition, our framework helps end-users without technical background to compose services. Our framework understands user intents by automatically mining tasks and constraints from end-users' goals. We incorporate the constraints specified in end-users' goals to select services. Our case studies show that our framework can achieve a high precision and a high recall to identify tasks and constraints from textual descriptions in natural languages. Our framework is effective in identifying optimal services for end-users. The results of our user study demonstrate that the recruited end-users in our study find our framework highly useful.

In future work, we plan to enhance the automation level of our framework. We can automatically retrieve How-to instructions based on the selection history. We can also remove infrequently performed tasks for task models. Moreover, we would like to perform a larger case study including more How-to instructions and evaluators to remove possible bias.

References

[1] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, "Open information extraction for the web," in *IJCAI*, vol. 7, 2007, pp. 2670–2676.

[2] D. L. Chen and R. J. Mooney, "Learning to interpret natural language navigation instructions from observations," in *Proc. 25th AAAI Conf. Artificial Intell.*, 2011, pp. 859–865.

[3] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Unsupervised named-entity extraction from the web: An experimental study," *Artificial Intell.*, vol. 165, no. 1, pp. 91–134, 2005.

[4] C. Fellbaum, *WordNet*. Wiley Online Library, 1998.

[5] F. Hang and L. Zhao, "Developing a meta-model to support end-user in composition," in *Int'l. Conf. Web Services*, pp. 471–478.

[6] T. Hornung, A. Koschmider, and G. Lausen, "Recommendation based process modeling support: Method and user experience," in *Conceptual Modeling*. Springer, 2008, pp. 265–278.

[7] J.-M. Jézéquel, B. Combemale, O. Barais, M. Monperrus, and F. Fouquet, "Mashup of metalanguages and its implementation in the kermeta language workbench," *Software & Systems Modeling*, vol. 14, no. 2, pp. 905–920, 2015.

[8] B. Katz, "From sentence processing to information access on the world wide web," in *AAAI Spring Symp. Natural Language Process. for the World Wide Web*, vol. 1, 1997, p. 997.

[9] D. Klein and C. D. Manning, "Accurate unlexicalized parsing," in *Proc. 41st Annual Meeting on Assoc. for Computational Linguistics-Volume 1*, 2003, pp. 423–430.

[10] C. Kwok, O. Etzioni, and D. S. Weld, "Scaling question answering to the web," *ACM Trans. Inform. Syst.*, vol. 19, no. 3, pp. 242–262, 2001.

[11] T. Lin, P. Pantel, M. Gamon, A. Kannan, and A. Fuxman, "Active objects: Actions for entity-centric search," in *Proc. 21st Int'l. Conf. WWW*, 2012, pp. 589–598.

[12] B. Liu, C. W. Chin, and H. T. Ng, "Mining topic-specific concepts and definitions on the web," in *Proc. 12th Int'l. Conf. WWW*. ACM, 2003, pp. 251–260.

[13] H. Liu and P. Singh, "Conceptnet practical commonsense reasoning tool-kit," *BT technology J.*, vol. 22, no. 4, pp. 211–226, 2004.

[14] X. Liu, Y. Hui, W. Sun, and H. Liang, "Towards service composition based on mashup," in *IEEE Congress on Services*, 2007, pp. 332–339.

[15] C. D. Manning, P. Raghavan, H. Schütze *et al.*, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008.

[16] Z. Maraikar, A. Lazovik, and F. Arbab, "Building mashups for the enterprise with sabre," in *Service-Oriented Computing–ICSOC*. Springer, 2008, pp. 70–83.

[17] M. Matera, M. Picozzi, M. Pini, and M. Tonazzo, "Peudom: A mashup platform for the end user development of common information spaces," in *Web Engineering*. Springer, 2013, pp. 494–497.

[18] A. Moustafa and M. Zhang, "Multi-objective service composition using reinforcement learning," in *Service-Oriented Computing*. Springer, 2013, pp. 298–312.

[19] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[20] L. Richardson and S. Ruby, *RESTful web services*. O'Reilly Media, Inc., 2008.

[21] M. Shamsfard and A. A. Barforoush, "Learning ontologies from natural language texts," *Int'l. J. of human-computer studies*, vol. 60, no. 1, pp. 17–63, 2004.

[22] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz, "Analysis of a very large web search engine query log," in *SIGIR Forum*, vol. 33, no. 1. ACM, 1999, pp. 6–12.

[23] J. Thomason, S. Zhang, R. Mooney, and P. Stone, "Learning to interpret natural language commands through human-robot dialog," in *Proc. 2015 Int'l. Joint Conf. Artificial Intell. (IJCAI)*, 2011, pp. 859–865.

[24] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proc. Conf. North American Chapter of the Assoc. Computational Linguistics on Human Language Technology-Volume 1*, 2003, pp. 173–180.

[25] B. Upadhyaya, Y. Zou, S. Wang, and J. Ng, "Automatically composing services by mining process knowledge from the web," in *Service-Oriented Computing*. Springer, 2013, pp. 267–282.

[26] G. Wang, S. Yang, and Y. Han, "Mashroom: end-user mashup programming using nested tables," in *Proc. 18th Int'l. Conf. WWW*. ACM, 2009, pp. 861–870.

[27] H. Xiao, Y. Zou, R. Tang, J. Ng, and L. Nigul, "Ontology-driven service composition for end-users," *Service Oriented Computing and Application*, vol. 5, no. 3, pp. 159–181, 2011.

[28] J. Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding mashup development," *IEEE Internet Computing*, vol. 12, no. 5, pp. 44–52, 2008.

[29] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, 2004.