Context-aware Generic Service Discovery and Service Composition

Yifei Zhang, Jianing Wang, Yuhong Yan
Concordia University, Montreal, Canada
Email: yuhong@encs.concordia.ca

Abstract—Service Composition is to create a new business process by composing several services in order to fulfill business goals that individual services cannot achieve. Service discovery and service composition can be highly adaptive to contexts, i.e., according to context information, e.g., location, budget and time, we can discover and compose these services to satisfy particular requirements in the contexts. Moreover, we want to contain non-electronic services, e.g., restaurants, into service composition. These services are not considered in existing service composition research, but are ubiquitous in mobile phone working environment. In this paper, we present the methods of discovering and composing non-electronic services based on contexts. We build a constraint-based context model which is more suitable for service composition algorithm than the other context model. Our service composition algorithm uses soft constraints. With this feature, the service composition algorithm can give the user several “good enough” solutions, instead of null solution. More importantly, we include non-electronic services into service composition. Throughout the paper, an entertainment planner is used as a motivated example.

I. INTRODUCTION

The goal of Services Computing is to enable IT services and computing technology to perform business services more efficiently and effectively [7]. Automated Service Composition as an approach of service creation has drawn a lot of attentions. Most of these researches focus only the electronic services, i.e., automatic services that are provided by software systems. However, the scope of Service Computing should cover all kinds of services, including both electronic and non-electronic services, just like what UDDI [9] can list1. In this paper, our goal is to explore the methods to discover and compose the generic services.

A mobile phone user is a very good example of using generic services such as restaurants, retail stores etc., not only particularly electronic services, for his/her daily life. Another feature of this environment is that it should be context-based. Context information, e.g., location, identity, and time, should be used as a part of requirements for service discovery and service composition, in addition to other business goals. Yet another challenge is that mobile devices have limited computation resources, various screen sizes, and limited speed of Internet access. Therefore, the user interface and the composition algorithm using on the mobile devices should consider these limitations.

Our main contributions in this paper include:

1) We propose a constraint-based context model. In order to ease knowledge sharing and reuse, we firstly build an ontology for context description. A part of the ontology is domain related and can be extended for different domains, if not currently modeled. Then, a constraint-based context model is built on the top of the ontology. This model is able to deal with both propositions and real values. Using our context model, we can express both the current context and the business goals.

2) We extend the services in service composition into non-electronic services such that service composition can be performed on all kinds of services (generic services as the paper title states). We present a way to discover and compose the non-electronic services. The preconditions and the post conditions of the services are modeled as contexts and constraints determine the execution order of the services in a composite business process.

3) We develop a service composition algorithm features soft constraints, dissatisfaction of which does not invalidate a plan, but devalue it. With this feature, the service composition algorithm can give the user several “good enough” solutions, instead of null solution. The system architecture is designed to generate minimal data traffic between the service and the mobile device.

We build an entertainment planner as an example to demonstrate the procedure of context-aware service discovery and service composition over mobile web apps.

This paper is organized as follows. After the introduction in this section, we present a motivation example. In section III, we introduce the related definitions in constraint model for context representation. We discuss composed service discovery and mashup in Section IV. Section V includes the details related to algorithm of Web service composition. Section VI describes the implementation of the entertainment planner. At the last, we compare our work with the existing literatures (Sec. VII) and draw a conclusion (Sec. VIII).

1UDDI covers all kinds of services in all kinds of business using a category system e.g., North America Industry Classification System (NAICS).
II. A MOTIVATING EXAMPLE AND THE CHALLENGES

Most smart phone users need applications to help their daily activities. One scenario interested us is as follows. When you travel to a new city for a business trip, you would most likely want to get some entertainment in the evening after a long day of work. We design an application called Personal Entertainment Planner to collect the users' interests, discover nearby related services, and make a plan in the form of business process for the user. Suppose you can input the time period, e.g., from 7:00 PM to 11:00 PM this evening, location, e.g., “within 2 km of my location”, budget, e.g., $50. With the above information plus other information the mobile web app can access, such as contacts, personal preferences, the Entertainment Planner can discover the services according to those constraints, and is expected to give the following options (with real business names in Montreal):

1) Plan 1: dinner at Restaurant L’Autre Saison from 7:00 - 8:00; Watching movie “The help” at cinema “Cinema Banque Scotia Montreal” from 8:45-11:00;
2) Plan 2: Dinner at Seven Night Club and watch the Hockey game “Canadiens vs. Boston Bruins” from 7:30 to 11:00;

The entertainment planner benefits from context information, such as location, identity, and time, which is convenient to collect on mobile web app. The difficulty is how to use the context in service composition. Planning algorithms need a formal model of the context information. We need to define operations over the contexts so that the system states can be transferred over these operations. Especially when the context variables have infinite domains, some composition algorithms cannot handle these domains.

Most services for entertainment are not online services like restaurants, movie theaters, and bars. UDDI [9] or the other industrial e-commerce standards indeed cover these non-electronic services. Although these services should be included in the service computing study, none has ever used them in composing business processes. We need a model for these services so that the composition algorithm can use them. Another problem is how to discover these services. UDDI is not an option because public UDDI servers are practically unavailable. We need to use normal service engine or service index site to discover these services.

III. MODELS FOR CONTEXTS AND SERVICES

A. Context Model

Dey defines context as “any information that can be used to characterize the situation of an entity(user)” [2]. For the mobile application, context is the information automatically detected by the device, e.g., the location, the identity, and the current time, or provided by the user e.g., the budget and the range of moving.

There exist several context models, including key-value models, markup schema models, object oriented models, graph models, ontology models, and logic models [16]. In our work, we expect our context models can handle equality constraints and inequality constraints over continuous, discrete, and boolean variables. Also, the operations over the contexts are expected to be handled by a compositional and optimized algorithm. However, no existing context models can do this. Therefore, we propose our own constraint model for our context representation.

For modeling the problem precisely, we give an ontology as in Table 1. Most of the concepts in this ontology are domain independent, except the Service type. People can add new service types according to their domain.

### Table 1

<table>
<thead>
<tr>
<th>type</th>
<th>subtyping</th>
<th>sub-sub-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location</td>
<td>Current Location</td>
<td></td>
</tr>
<tr>
<td>Destination</td>
<td>Start Location</td>
<td></td>
</tr>
<tr>
<td>Distance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>TimePoint</td>
<td>StartTimePoint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EndTimePoint</td>
</tr>
<tr>
<td>Distance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Money</td>
<td>Cost</td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td>Movie</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Restaurant</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shopping</td>
<td></td>
</tr>
<tr>
<td>Self-Service</td>
<td>Driving</td>
<td></td>
</tr>
<tr>
<td>UserPreference</td>
<td>Default</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TimeFirst</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CostFirst</td>
<td></td>
</tr>
<tr>
<td>Indicator</td>
<td>Soft Indicator</td>
<td>TimeConnect</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TimeCost</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MoneyCost</td>
</tr>
<tr>
<td>Hard Indicator</td>
<td>MovieCHK</td>
<td>DiningCHK</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ShoppingCHK</td>
</tr>
</tbody>
</table>

Based on the ontology, we define the variables and the constraints.

**Definition 1**: A variable is a tuple \((\text{varName}, \text{dataType}, \text{ontologyType})\).

In Definition 1, \text{varName} is a symbol to represent a variable. \text{dataType} := real | naturenumber | integer | boolean | string. \text{ontologyType} is a type in Table I for the semantic meaning of the variable. All variables used in this paper are listed in Table II.

**Definition 2**: A constraint is represented as an equality or an inequality of variables.

There are two kinds of constraint, one is hard constraint, the other one is soft constraint. Hard constraints defines prohibit regions of variable assignments. They are the constraints that must be satisfied. A soft constraint merely imposes a penalty on certain assignments rather than prohibiting them [14]. Each soft constraint is assigned a penalty value between 0 and 1. 1 means the constraint is
Table II

<table>
<thead>
<tr>
<th>variableName</th>
<th>dataType</th>
<th>ontologyType</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>string</td>
<td>Location</td>
</tr>
<tr>
<td>distance</td>
<td>real</td>
<td>Distance</td>
</tr>
<tr>
<td>money</td>
<td>number</td>
<td>Cost</td>
</tr>
<tr>
<td>time</td>
<td>real</td>
<td>TimePoint</td>
</tr>
<tr>
<td>duration</td>
<td>real</td>
<td>Duration</td>
</tr>
<tr>
<td>startTimeP</td>
<td>real</td>
<td>StartTimePoint</td>
</tr>
<tr>
<td>endTime</td>
<td>real</td>
<td>EndTimePoint</td>
</tr>
<tr>
<td>totalMoneyCost</td>
<td>number</td>
<td>Cost</td>
</tr>
<tr>
<td>serviceActivity</td>
<td>real</td>
<td>Service</td>
</tr>
<tr>
<td>direction</td>
<td>real</td>
<td>Driving</td>
</tr>
<tr>
<td>up</td>
<td>real</td>
<td>UserPreference</td>
</tr>
<tr>
<td>connectConstraint</td>
<td>real</td>
<td>TimeConnect</td>
</tr>
<tr>
<td>globalConnect</td>
<td>real</td>
<td>TimeConnect</td>
</tr>
<tr>
<td>costConstraint</td>
<td>real</td>
<td>TimeCost</td>
</tr>
<tr>
<td>mv</td>
<td>boolean</td>
<td>MovieCHK</td>
</tr>
<tr>
<td>dv</td>
<td>boolean</td>
<td>DiningCHK</td>
</tr>
<tr>
<td>sv</td>
<td>boolean</td>
<td>ShoppingCHK</td>
</tr>
</tbody>
</table>

B. Service Model

A service can be applied to a context if its execution conditions are satisfied. In service composition, we make the context as a description of a system state. Applying a service on a context can move system from one state to another.

Definition 4: A service \( a \) is a tuple \( \langle \text{Pre}_a, \text{Attr}_a, P_a \rangle \), where \( \text{Pre}_a \) is a finite set of preconditions and \( P_a \) is a finite set of effects. \( \text{Attr}_a \) is the attributes of \( a \). \( \text{Pre}_a \) and \( P_a \) contain hard constraints.

Duration is one of the attributes of services. A duration is a time interval with a start time point and an end time point. We use a variable serviceConnect \( sc \) to describe how well two services are connected.

Definition 5: Assume two services \( a_1 \) and \( a_2 \) has durations \( X = [s_1, e_1] \) and \( Y = [s_2, e_2] \) respectively. The variable serviceConnect \( sc \) is calculated using Equation 3, representing the degree of time intervals connection between the two services.

Allen’s interval algebra [1] defines possible relations between time intervals (Figure 1). Without loss of generality, we consider \( s_1 \geq s_2 \) and the set of relations \( \{ X < Y, X m Y, Y s X, Y f X, X = Y \} \). \( offset_{con} \) is the difference between \( e_1 \) and \( s_2 \) (Equation 1). \( sc \in [0, 1] \) is a measure of how the two services are connection. \( sc = 1 \) means \( a_1 \) “meets” \( a_2 \), a preferred case. \( 0 < sc < 1 \) means \( a_1 \) overlaps with \( a_2 \) and \( a_2 \) finishes before \( a_2 \), a feasible case. We prefer the \( sc \) value the higher the better, because this means the overlapping is less. \( sc = 0 \) means \( a_1 \) is “equal” to \( a_2 \), or \( a_2 \) is “during” to \( a_1 \), an infeasible case.

\[

offset_{con} = \begin{cases} 
\frac{e_2 - s_1}{e_1 - s_2} & \text{if } (e_1 \leq s_2) \\
\frac{e_1 - s_2}{e_1 - s_2} & \text{if } (s_1 < e_2)
\end{cases}
\]  

\[

s = 1 - \frac{offset_{con}}{e_2 - s_2} \tag{1}
\]

\[

sc = \begin{cases} 
0 & \text{if } (e_1 < e_2) \\
\frac{s}{offset_{con}} & \text{if } (e_1 < e_2)
\end{cases} \tag{2}
\]

Figure 1. Possible relations between two intervals [17]

We can apply a service to a context if the context satisfied some conditions defined in Definition 6.

Definition 6: Assume \( X \) is a context. A service \( a \) is applicable to \( X \), denoted as \( X \rightarrow a \), if \( X \) satisfies \( \text{Pre}_a \):

- If we apply a service \( a \) to a context \( X \), the context will be changed to \( X' \). We can calculate \( X' \) as the following.

Definition 7: Assume a service \( a \) is applicable to a context \( X \), \( X \rightarrow a \). A new context \( X' \) is transformed from the context \( X \) after applying \( a \) is denoted as \( X \xrightarrow{a} X' \). The context \( X' \) applies the following assignments for variables.
• location' = La, where location' is the location of X', and La is the destination of a;
• totalMoneyCost' = Cx + Ca, where totalMoneyCost' and Cx are the cost to reach X' and X separately, and Ca is the cost of a;
• startTimeP' = Tsa, where startTimeP' is the planned start time of X', and Tsa is the start time of a;
• startTime' = Tcx, where startTime' is the planned start time of X', and Tcx is the end time of X;
• endTime' = Tca, where endTime' is the planned end time of X', and Tca is the end time of a;
• duration' = endTime' - startTime', where duration' is the time cost of X';
• totalTimeCost' = Dx' + TCx, where totalTimeCost' and TCx are the total time cost of X' and X separately, and Dx' is the time cost of X';
• if a is a Restaurant, X'.dv = true; if a is a Movie, X'.sv = true;

C. Context Evaluation

Based on the definition of Constraint and Context, we can use constraints to evaluate one context. Those constraints include hard constraints and soft constraints. Hard constraints are those which we definitely want to be true. These might relate to the successful assembly of a mechanism. Soft constraints are those we would like to be true - but not at the expense of the others. We can provide one soft constraint a value to represent its degree of satisfaction. If one soft constraint is satisfied completely, its degree of satisfaction is 1. Otherwise, we will calculate the penalty for its degree of satisfaction. The context X' in Definition 7 satisfies the following constraints.

• c = \{|X'.location - X_i.location| \leq 2000\}. This means that the distance between one activity’s location and user’s current location cannot exceed 2000 meters. c should always be true. This is a hard constraint. X_i is the initial context.
• timeConstraint' = tc(TC_x', T_g), where timeConstraint' represent the degree of the time budget satisfaction from initial context to the context X'. TC_x' is the totalTimeCost of X', and T_g is the time budget. offsettc is the difference between TC_x' and T_g. This a soft constraint, we use function tc to calculate the penalty and its value. As timeConstraint' grows larger from 0 to 1, the time usage will be closer to the time budget. timeConstraint' = 0 means the time usage is 0 or far beyond the time budget. timeConstraint' = 1 means the time usage meets the time budget perfectly.

\[
offsettc = \begin{cases} 
T_g - TC_x' & \text{if } (TC_x' \leq T_g) \\
TC_x' - T_g & \text{if } (T_g < TC_x') 
\end{cases}
\] (4)

\[
tc = 1 - \frac{offsettc}{T_g}
\] (5)

\[
tc(TC_x', T_g) = \begin{cases} 
tc & \text{if } (offsettc \leq T_g) \\
0 & \text{otherwise}
\end{cases}
\] (6)

• costConstraint' = cc(MC_x', C_g), where costConstraint' represent the degree of the budget satisfaction from initial context to the context X'. MC_x' is the totalMoneyCost of X', and C_g is the budget. offsetcc is the difference between MC_x' and C_g. This a soft constraint, we use function cc to calculate the penalty and its value. As costConstraint' grows larger from 0 to 1, the money usage will be closer to the budget. costConstraint' = 0 means the money usage is 0 or far beyond the budget. costConstraint' = 1 means the money usage meets the budget perfectly.

\[
offsetcc = \begin{cases} 
C_g - MC_x' & \text{if } (MC_x' \leq C_g) \\
MC_x' - C_g & \text{if } (C_g < MC_x') 
\end{cases}
\] (7)

\[
cc = 1 - \frac{offsetcc}{C_g}
\] (8)

\[
cc(MC_x', C_g) = \begin{cases} 
cc & \text{if } (offsetcc \leq C_g) \\
0 & \text{otherwise}
\end{cases}
\] (9)

If one context X' is an initial context, the value of the connectConstraint in X' will be 1. If one context X' is not an initial context, it can be generated by using the procedure X' \texttt{a1} X' from Definition 7. Then, we can extend that procedure to be X' \texttt{a2} X'. If the context X is not an initial context, a1 will be a normal service. If the context X is an initial context, in order to keep the correctness of the new procedure a1 will be one special service just for the initial context. After that, the connectConstraint of X' can be calculated as the following.

Definition 8: The property connectConstraint in the context X' represents the degree of time intervals connection between two services e.g., a1, a2. Based on Definition 5 and 7, we can calculate connectConstraint by using Equation 11 and 12. t_e is the end time of the context X', t_s is the start time of the context X' and t_sp is the planned start time of the context X'. This a soft constraint, we use function xconCAL to calculate the penalty and its value.

\[
offsetxcon = \begin{cases} 
t_s - t_sp & \text{if } (t_sp \leq t_s) \\
t_sp - t_s & \text{if } (t_s < t_sp) 
\end{cases}
\] (10)

\[
s = 1 - \frac{offsetxcon}{t_e - t_sp}
\] (11)

\[
xconCAL(t_e, t_s, t_sp) = \begin{cases} 
s & \text{if } (t_s < t_e \land offsetxcon < t_e - t_sp) \\
0 & \text{otherwise}
\end{cases}
\] (12)

Based on Definition 5, in Equation 1, 2 and 3, e_2 represents the end time of service a_2, e_3 represents the start time of service a_2 and e_1 represents the end time of a_1.
Based on Definition 7, we know that the end time of service \( a_2 \) is the end time of context \( X' \), the start time of service \( a_1 \) is the planned start time of context \( X' \). The end time of service \( a_1 \) is the end time of context \( X \). At the same time, the end time of context \( X \) is the start time of context \( X' \). So, we can convert the Equation 1, 2 and 3 to the Equation 10, 11 and 12 separately.

**Definition 9:** \( \text{globalConnect}' \) is the value of globalConnect in Context \( X' \) and \( cc' \) is the value of connectConstraint in Context \( X' \). \( gc \) is the value of globalConnect in context \( X \).

\[
globalConnect' = \begin{cases} gc & \text{if } (cc' \geq gc) \\ cc' & \text{if } (cc' < gc) \end{cases} \tag{13}
\]

We can use a property to quantify one context because each context can be evaluated by using soft constraints. Soft constraints are assigned with a numeric value from 0 to 1. On the other hand, hard indicators for one context can also be associated with numeric values: 0 and 1, e.g., MovieCHK, DiningCHK, ShoppingCHK. For MovieCHK, 1 means a user has watched a movie. In other word, these hard indicators are special soft constraints. So, We can define a variable to represent the constraints of one context as following.

**Definition 10:** A \( \omega \) for one context \( c \) is a tuple \((hardValue, cc, mc, tc)\), where

- \( hardValue = ve + dc + sc \)
- \( ve, dc, sc \) are the numeric values of hard indicators(MovieCHK, DiningCHK, ShoppingCHK) respectively;
- \( cc \) is the value of one context's constraint globalConnect;
- \( mc \) is the value of one context's cost Constraint;
- \( tc \) is the value of one context's soft constraint timeConstraint;

**Definition 11:** The value of \( \omega \) for one context \( \omega_c \) can be calculated by using equation 14. Two contexts' \( \omega \) e.g., \( \omega_1, \omega_2 \) can be compared by using their values.

\[
\omega_c = hardValue + (cc + mc + tc)/3 \tag{14}
\]

IV. SERVICE DISCOVERY MASHUP

Based on the business goals, we can discover related services. In our research, we want to use non-electronic services in service composition. Therefore, we use general purpose search engines such as Google, Yelp or Foursquare to discover services.

The mashup accepts the user inputs and query online resources for discovering related services. The mashup extracts the service information from the query results and formalizes the services according to the service model in the previous section. The formalized services and the user inputs are fed to the service composition engine to calculate a plan.

As the current technical capacity allows, building a mashup is a manual task rather than automatic approach. In order to demonstrate the wide coverage of services, we use various online resources. We use Google place service, Yelp service and Foursquare service to search for business services, e.g., restaurant and shopping mall. Google Maps Web service is to find driving direction from the original place to the destination. Google show time as an HTML engine returns movie services.

**Google Places API** [5] is a RESTful service which allows you to query for place information on a variety of categories. You can search for places either by proximity or a text string.

**Yelp v2.0 API** [18] is a RESTful service which enables access to more relevant search results that more closely match the results on Yelp. It uses a standard and secure authorization protocol (OAuth 1.0a, xAuth).

**Foursquare Search Venues API** [3] is a RESTful service which Returns a list of venues near the current location, optionally matching a search term on Foursquare. User will need a client ID and client secret to make a useless venue search or explore request.

**Google Maps Web Services** [4] is a RESTful service. The output of the service is in either JSON or XML. For example, the query for get JSON output is

https://maps.googleapis.com/maps/api/directions/json?origin=51.5087,-0.1208&destination=51.5187,-0.1408&sensor=false

The query above is for searching driving directions between two places (Geographic position 51.5087,-0.1208 and 51.5187,-0.1408). The Directions API can return multi-part directions using a series of waypoints, durations and distance.

**Google Show Time** is a part of Google search engine. You can send an HTTP query like

http://www.google.com/movies?near=45.496330,-73.578829 to get movie schedule near the location you put in the query string. The returned response is in HTML format. We use Jsoup to locate the cinema address and shot times for movies from the HTML response.

The mashup architecture is shown in Figure 2. The outputs (latitude and longitude) from the business service integration module and the Google movie show time are fed into Google map as inputs for directions.

The mashup is also responsible for extracting the information from the responses and formalizing the services using the models in the previous section. It is straightforward to extract service attributes, like duration, address, and starting time for movies. When some values are unavailable, we have predefined the rules to set the value. For example, the cost for a restaurant is $20 and the cost for a movie is $10.

V. SERVICE COMPOSITION

A. Problem Description

A system state \( s \) represents a current context of the system.
Definition 12: A system state \( s \) is a context, i.e., a set of variable assignments.

For example, the initial state \( s_0 \) for the motivation example in Section III can be described as below:

\[
\begin{align*}
\{ mv = \text{false, } dv = \text{false, } sv = \text{false, } \\
\text{money} = 100, \text{location} = \text{“H3K2S5”} \}
\end{align*}
\]

(15)

Definition 13: Suppose there are two states \( s_1, s_2, \omega_1 \) is the value of \( s_1 \) and \( \omega_2 \) is the value of \( s_2 \). If \( \omega_1 \) is larger than \( \omega_2 \), we can say \( s_1 \) is better than \( s_2 \).

Definition 14: A service composition query is a tuple \( \langle s_g, s_0, C \rangle \), where

- \( s_g \) is a target state;
- \( s_0 \) is an initial state;
- \( C \) is a set of constraints satisfied at any time;

For a service composition query, \( s_g \) contains both a set of expected business targets that user want to achieve and a set of constraints on the duration, location and cost. \( C \) is the constraints for any state. e.g.,

\[
\begin{align*}
\{ mv = \text{true, } dv = \text{true, } sv = \text{true, } \\
\text{money} = 20, \text{location} = \text{“H3K2S5”} \} \\
C = \{ |\text{location} - \text{“H3K2S5”}| \leq 2000 \}
\end{align*}
\]

(16)

Definition 15: The state transition function \( \gamma \) of one service \( a = \langle Pre_a, Attr_a, P_a \rangle \) for any state \( s \) is \( \gamma(s, a) = s' \), iff \( a \) is applicable to \( s \), i.e., \( s \gg a \).

Based on the definitions above, we now define the problem of service composition.

Definition 16: A service composition problem is a tuple \( \langle s_g, s_0, A, C \rangle \), where

- \( s_g \) is a goal state;
- \( s_0 \) is an initial state;
- \( A \) is a set of available services;
- \( C \) is a set of constraints satisfied at any time;

For a set of available services \( A \), a service composition problem is to produce a business procedure that can generates one or several states which should be better than \( s_g \) from the initial state \( s_0 \) provided that all the constraints in \( C \) are satisfied during the composition process.

Definition 17: A solution \( \pi \) to the service composition query \( \langle s_g, A, s_0, C \rangle \) is a sequence of activities \( \langle \pi_1; \ldots; \pi_n \rangle \), in which each \( \pi_i (i \in [1, n]) \) is a set of paralleled actions. \( \pi_1 \) is applicable to \( s_0 \). \( \pi_i \) is applicable to \( \gamma(s_i-2, \pi_{i-1}) \) when \( i \in [2, n] \). \( s_t \) hold at a target state \( s_t = \gamma(\ldots(\gamma(s_0, \pi_1), \pi_2)\ldots \pi_n) \). \( s_t \) should be better than \( s_g \). \( C \) are satisfied at any state \( s_t \), \( i \in [0, n] \).

B. composeAlgorithm

Now, we are looking for an algorithm to solve the problem. This service composition problem is not only a planning problem but also a constraint satisfaction optimization problem [15]. As we discussed above, several constraints in this problem are soft constraints. So, this composition problem can be saw as a soft constraint satisfaction problem. Branch and Bound is an efficient way to solve that kind of problem [6] [8].

Thus, we use a branch and bound algorithm as our planning algorithm. On the one hand, branch and bound algorithm uses Depth-First search to build its search tree which can includes all the possibilities of the problem. On the other hand, this algorithm can optimize the solutions. A general branch and bound method has two basic stages.

- Branching: splitting the problem into subproblems;
- Bounding: calculating lower or upper bounds for the objective function value of the subproblem;

Our goal is to find a certain number of good enough solutions for the problem. Based on the Definition 17, the goal can also be saw as to generate a certain number of states which are better than or equal to the goal state. Therefore, we use \( \omega \) for states to build bounds and branching because one \( \omega \) value can represent the quantitative value of one state, and it is comparable.

Definition 18: A lower bound \( LB \) contains current state’s \( \omega \). Each state has a lower bound.

Definition 19: An upper bound \( UB \) contains the \( \omega \) for one of the states which are better than the goal state. Each upper bound has a corresponding state.

We build a priority Queue \( UBQ \) to store the upper bounds. The size of \( UBQ \) is the same as the quantity of solutions.

Definition 20: An upper bound \( UB_g \) contains the \( \omega \) for the goal state.

Definition 21: An upper bound \( UB_{basic} \) only contains the hardValue of the goal state’s \( \omega \). The other three properties of \( UB_{basic} \) is 0. We use \( UB_{basic} \) to initialize the \( UBQ \).
The algorithm has four parameters: \( s \), the current state; \( UBQ \), a priority Queue to store the upper bounds; \( seq \), \( seq \) represents the current level of the search tree and \( seq \) is an integer; \( A \), a set of available services.

Algorithm 1 presents the detailed steps of branch and bound algorithm. Branch and bound algorithm starts from the initial state \( s_0 \). Line 1 defines a flag variable \( res \). In lines 2 to 4, we compare the smallest \( UB \) in \( UBQ \) with \( UB_g \). If \( UB \) is larger than \( UB_g \) (\( res \leftarrow \text{"default"} \)), this situation means we have found enough states which are better than the goal state and the process will be finished. If \( res \neq \text{"finish"} \), we will compare \( cc \) in \( s.LB \) with every upper bound in \( UBQ \) (lines 5 to 12). If \( cc \) in \( s.LB \) is larger than an arbitrary upper bound’s \( cc \) in \( UBQ \) (\( res \leftarrow \text{"candidate"} \)), this state can be recorded as a candidate. Otherwise, we will prune this state node and its subtree. After that, if \( res \leftarrow \text{"candidate"} \), we will check if the \( LB \) of this state can be inserted into the \( UBQ \) or not (lines 13 to 17). If so (\( res \leftarrow \text{"target"} \)), this state’s \( LB \) will be one of upper bounds and we will prune the subtree of this state node. If the current state \( s \) is just a candidate (line 18 \( res \leftarrow \text{"candidate"} \)), this means its childStates still have the possibility to be solutions. \( ActFilter \) (Line 19) is used to retrieve the available services from \( A \) depend on the state’s hard context-constraints and Constraint C. After that, we use \( service \) in \( AvailableServices \) and \( \gamma \) to generate new state \( s' \) (line 21). Then, we call the BnB procedure again to run the next iteration (line 22). If the current state is not a candidate, the process will be finished and return (line 25).

Algorithm 1 terminates after \( k \) solutions are reported or no states to expand. To reduce the space requirement, our algorithm uses depth-first search (DFS) as the search strategy. However, using DFS has possibilities to miss the best solution because DFS always selects the most recently generated node or deepest node to expend next [19]. For instance, there is more than one goal node in the tree we build, and our search decided to first expand the first subtree of the root where there is a solution at a very deep level of this sub tree, at the same time the other one sub tree of the root has the best solution, here comes the non-optimality of DFS that it is not guaranteed that the first goal to find is the optimal one, so we can conclude that DFS is not optimal [12].

\[ \text{Theorem 1: The time complexity of DFS branch-and-bound algorithm is exponential to the branching factor } b \text{ and the maximum depth } m. \]

\[ \text{Proof: } b \text{ is the branching factor and } m \text{ is the maximum depth of any path in the search tree. The depth-first search is asymptotically optimal because most states will not have a child-state which has the same } \omega \text{ value. Hence, the expected number of states expanded by depth-first search for finding several optimal states of a tree } T(b, m), \text{ as } m \to \infty, \text{ is } \theta(\beta^m), \text{ where } \beta \text{ is a constant, } 1 < \beta < b \text{ [19].} \]

VI. IMPLEMENTATION AND EXPERIMENTS

All development for the entertainment planner is done in the Eclipse Integrated Development Environment (IDE) on Windows 7 machines. Testing of the application was done on the Samsung galaxy S4 and the Samsung galaxy S2 under Wi-Fi and 3G network respectively. In this test, the download speed of WIFI environment is up to 15 Mbps and the download speed of 3G network is up to 5 Mbps.

We use the following scenario to test our entertainment planner. Supposing An user is in Montreal; amount of budget = $50; location = H3K1Z8; travel distance is 2.5km; starting time = 1:00pm; end time = 5:00pm; Activities= Movie, Restaurant, Shopping. Figure 3 is the screenshot of planning results. The average execution time for the application under different network environments are listed in the Table IV.

(Receiving Time include the time cost for network delay, receiving data from service discovery)

<table>
<thead>
<tr>
<th>Time(s)</th>
<th>Galaxy S4 3G</th>
<th>Galaxy S2 3G</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Discovery</td>
<td>3.23 3.18</td>
<td>3.2 3.14</td>
</tr>
<tr>
<td>Receiving</td>
<td>0.39 0.44</td>
<td>0.47 0.42</td>
</tr>
<tr>
<td>Service Composition</td>
<td>3.5 4.2</td>
<td>4.0 4.6</td>
</tr>
</tbody>
</table>
VII. RELATED WORK

RESTful services make the system become resource-oriented. It is necessary to compare RESTful service composition and RESTful service mashups. RESTful services can also composed into business processes. The composition of RESTful service focuses on the resource composition and "state transfer". Compared to RESTful service composition, a mashup is restricted at the data-level integration, and most uses of RESTful services in mashup are limited to fetching data from remote resources. It usually does not involve updating or manipulating remote data sources or other resources [20]. JOpera [10] is one of the most mature platforms for supporting RESTful service composition. JOpera provides a visual language for defining a control flow and a data flow transfer graph, as well as an execution engine for the resulting workflow. Nodes in the control flow graph present tasks that are dynamically bound to adapters such as local UNIX programs, service invocation, etc., and "glue" adapters that perform local computations (e.g., XPath queries, XSLT transformations, etc.). The composition results can be described in an BPEL extension for REST [11]. The composite service can be provided as a new RESTful service.

VIII. CONCLUSION

In this paper, we demonstrate the ability of mobile web application to discovery related services using general purpose search engine according to context information and study the context-aware service composition problem. The discovered services, including non-electronic services, are composed into business processes by the composition engine. The composition algorithm with soft constraints can give the user some "good enough" solutions, instead of null solution. The next step of our work is to add a Re-planning function and a user requirements adaptive function into this service composition algorithm.

REFERENCES


