O2O Service Composition with Social Collaboration

Wenyi Qian*†, Xin Peng*†, Jun Sun‡, Yijun Yu§, Bashar Nuseibeh§¶, and Wenyun Zhao*†
*School of Computer Science, Fudan University, Shanghai, China
†Shanghai Key Laboratory of Data Science, Fudan University, China
‡Singapore University of Technology and Design, Singapore
§School of Computing and Communications, The Open University, UK
¶Lero - The Irish Software Research Centre, University of Limerick, Limerick, Ireland

Abstract—**In Online-to-Offline (O2O) commerce, customer services may need to be composed from online and offline services. Such composition is challenging, as it requires effective selection of appropriate services that, in turn, support optimal combination of both online and offline services. In this paper, we address this challenge by proposing an approach to O2O service composition which combines offline route planning and social collaboration to optimize service selection. We frame general O2O service composition problems using timed automata and propose an optimization procedure that incorporates: (1) a Markov Chain Monte Carlo (MCMC) algorithm to stochastically select a concrete composite service, and (2) a model checking approach to searching for an optimal collaboration plan with the lowest cost given certain time constraint. Our procedure has been evaluated using the simulation of a rich scenario on effectiveness and scalability.**

I. Introduction

With the growing number of alternative Web services that provide equivalent functionality but differ in Quality of Service (QoS), complex applications are increasingly constructed by service composition, automating business processes by composing exisiting component services [1], [5]. Existing research on service composition focuses on selecting an optimal set of component services from given user's requirements and preferences by local selection [7], [19], global optimization [4], [5], [29], [30], or combining both [1]. This line of research considers online services only, that is, computational services invoked remotely via the Internet.

However, online services are increasingly related to offline facilities or social services involving human agents. For example, after reserving an archived document using an online service provided by a library, a user needs to physically collect the document. Bringing online consumers into real-world stores, "Online-to-Offline" (O2O) commerce has become a fast-growing area in eCommerce, with an estimated trillion dollar market [22], [11]. With the support of QR (Quick Response) codes, one can quickly access an online service, such as ordering a product associated with a physical world object by scanning its QR code using mobile phone, or consume a service in the physical world, such as having a dinner, with the QR code received on mobile phone as an electronic voucher.

Offline services take place in the physical space. They are bounded to certain physical locations and their executions may require the availability of certain physical objects. For example, a library provides a **Fetching Reserved Document** service at its service stations; a shop that provides a **Scanning Document** service requires that a document is taken to the shop for scanning. Therefore, the composition of online and offline services needs to consider not only the quality attributes and parameter passing in the virtual space but also the route planning and the passing of physical objects in the physical space. For example, it may be beneficial to choose a **Document Reservation** service that is more expensive but has a closer service station of the same library for fetching the reserved document.

On the other hand, the development of the emerging sharing economy (also known as collaborative consumption) [14] allows peer-to-peer sharing of accesses to goods and services through community-based online coordinations. For example, Amazon has developed an app called "On My Way" which would allow people to deliver packages *en route* to their destinations to cut cost while improving customer experience [17]. Through social collaboration, a user may recruit a group of people to participate in the execution of offline services, e.g., to fetch a reserved document from a library service station and take it to a shop for scanning, on the way to their own destinations. Such opportunistic collaboration is often cheaper than executing all the offline services by the user in person or by employing a normal delivery service.

A challenge to achieve the benefit of the proposed O2O service composition with social collaboration is to formally model both online and offline services and their interplay with the locations shared between mobile agents and their individual interests and concerns. In this paper, we propose an approach to O2O service composition which incorporates offline route planning and social collaboration into the considerations of service selection. We start with modeling O2O service composition using timed automata [2], which incorporates the concepts of abstract/concrete composite service, online/offline services, as well as agents (the user and candidate collaborators) and physical objects. These models define the O2O service composition precisely as an optimization problem. However, the complexity in the optimization is very high, that is, given an abstract O2O composite service, not only may there be many corresponding concrete composite services but also, for each concrete composite service, there may be many different ways in which agents can collaborate to reduce cost. Therefore, we propose practical algorithms to approximate the optimal solution. In particular, we propose a two-level procedure: (1) given an abstract composite service, we use a variant of the Markov Chain Monte Carlo (MCMC) [15] algorithm to select an instance for each online or offline abstract service so as to identify candidate concrete composite services; (2) for each candidate concrete composite service, we apply a model checker [6] to identify a collaboration plan with the lowest cost (including both cost of service consumption and human labor) that satisfies the given time constraint. We evaluate the proposed approach with a simulated scenario and our results support its effectiveness and scalability.

As far as we know, this is the first work on defining the problem of O2O service composition with social collaboration. The paper's novel contribution is therefore a formal model of the problem as well as an approach for solving it.

II. Motivating Example

Suppose that a user Bob would like to borrow an archived document from a library, get a scanned copy and print a bound copy of the document. The whole process can be represented by the composite service **Reproduce Borrowed Document** as shown in Figure 1, in which white and gray rectangles represent online and offline services respectively. The process includes two online services (i.e., **Document Reservation** and **Remote Printing**) and four offline services (i.e., **Fetching Reserved Document**, **Scanning Document**, **Fetching Bound Copy**, **Returning Document**).

To reproduce a borrowed document, a user such as Bob needs to first reserve the document using an online **Document Reservation** service provided by a library and will receive a QR code if the document is successfully reserved. Then the user needs to go to a selected library service station (a library can have multiple stations) and use the **Fetching Reserved Document** service to get the reserved document by scanning the QR code. Next he needs to take the document to a scanning service provider and use the **Scanning Document** service to get a scanned copy. After that he needs to upload the scanned copy and order a bound copy of the document using the online **Remote Printing** service provided by a printing service provider and will receive a QR code if the bound copy is successfully ordered and paid for. He can choose an offline service station of the printing service provider to use the **Fetching Bound Copy** service to get the bound copy by scanning the QR code. On the other hand, he needs to return the document to a service station of the library using the **Returning Document** service.

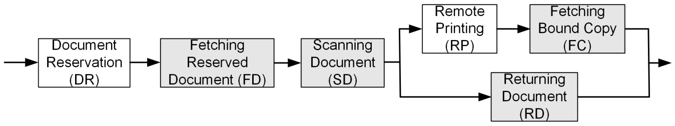


Fig. 1. Process of *Reproduce Borrowed Document*

Suppose that two libraries provide the **Document Reservation** service, each library has two offline service stations located at different places, which provide the **Fetching Reserved Document** and **Returning Document** services. Also consider three shops provide **Scanning Document** service; two companies provide **Remote Printing** service and each of the companies has two offline service stations providing the **Fetching Bound Copy** service. The QoS values (i.e., time and cost in this case) of each online or offline service instance are shown in Figure 2. Bob's initial location and the locations of offline service instances are also shown. The service instances in a bracket mean the preconditions of the service instance before the bracket. The lines between the locations represent segments on possible routes and we assume that the distance represented by each line is the same for simplicity.

To reproduce a borrowed document, Bob needs to instantiate the **Reproduce Borrowed Document** service by selecting an instance for each involved online or offline component service. According to service composition approaches, this problem can be considered as an optimization problem [1], in which the overall utility value has to be maximized while satisfying all global constraints. For the **Reproduce Borrowed Document** service, Bob wants to minimize the overall cost while satisfying the time constraint (i.e., within 100 time units). Given the QoS values shown in Figure 2, the best service composition is $\{DR_1, FD_1, SD_1, RD_1, RP_2, FC_4\}$ if we treat offline services the same as online services. Suppose Bob starts from his initial location L_0 and finally returns to it. According to this composition the best route for Bob to complete all the steps in person is: $L_0 \rightarrow L_1 \rightarrow L_3 \rightarrow L_5 \rightarrow L_9 \rightarrow L_{11} \rightarrow L_9 \rightarrow L_5 \rightarrow L_3 \rightarrow L_1 \rightarrow L_0$.

Obviously, there could be much better solutions when the distances between the locations of different offline services are considered. For example, if Bob chooses the service composition $\{DR_2, FD_4, SD_3, RD_4, RP_1, FC_1\}$, he can take a shorter route: $L_0 \rightarrow L_2 \rightarrow L_1 \rightarrow L_4 \rightarrow L_5 \rightarrow L_2 \rightarrow L_0$. This composition costs a little more on direct service consumption, but requires much less time and cost on the way. Therefore, O2O service composition needs to consider the locations of offline services and incorporate route planning on top of service selection.

The service composition for Bob can be further optimized by considering social collaboration. Suppose another person, Tom, is currently at the location L_3 and plans to walk to L_{10} via L_6 and L_7. He can wait at L_3 for the document to be reserved by Bob using DR_1 and then fetch the document using FD_1. Then he can take the document to L_6 for scanning using SD_2. Bob can get the scanned copy by email and order a remote printing using RP_2. Next Tom can go to L_7 to fetch the bound copy using FC_3.

Suppose another person, Jerry, will be at L_7 after Tom gets the bound copy and plan to walk to L_2 via L_8 and L_5. Jerry can get the document and bound copy from Tom at L_7 and return the document at L_8 using RD_2. When Jerry arrives at L_2, he can walk forward to L_0 to send the bound copy to Bob and then return to L_2, or Bob can walk to L_2 to get the bound copy and return to L_0. In either case, the cost on the way can be reduced greatly, as the additional distance that Bob, Tom and Jerry need to walk are only a round trip between L_0 and L_2. Furthermore, Tom and Jerry could be

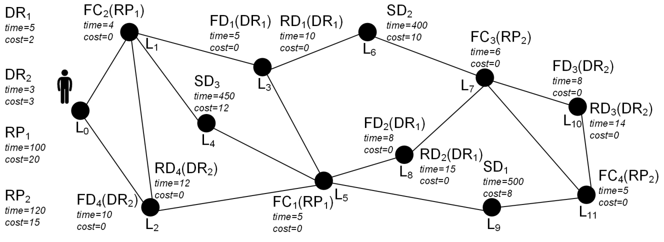


Fig. 2. QoS Values and Locations of Service Instances

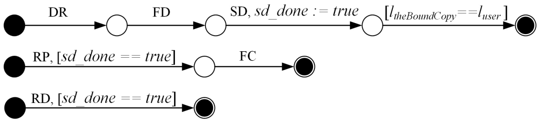


Fig. 3. FSM for Abstract Composite Service

potentially walking concurrently (to save overall time cost). We call such opportunistic delegation of services to potential collaborators as *social collaboration*, which can be challenging to incorporate into O2O service composition by considering the initial routes of social collaborators and computing an opportunistically beneficial plan for them to carry out the delegated tasks.

III. Service Model: Formalization

O2O service composition with social collaboration is complicated as it goes beyond traditional service composition. In particular, social collaboration requires the participation of multiple agents, which can be viewed as concurrent processes for carrying physical objects around. In the following, we present a general definition of the problem, which serves as the basis for developing approaches for solving the problem.

O2O Services An *abstract service* (e.g., **Document Reservation** in the motivating example) specifies the functionality of the service without referring to any concrete service instance. An *abstract composite service* (e.g., **Reproduce Borrowed Document**) specifies the compositional workflows using a set of abstract services for fulfilling the service requests. Without loss of generality, hereafter we assume that an abstract composite service AS is defined as a network of finite-state machines (FSM), i.e., the parallel composition of multiple finite-state machines where transitions are labeled with abstract services. For instance, the abstract composite service shown in Figure 1 can be easily translated to the network of FSM shown in Figure 3, where sd_done is an auxiliary Boolean variable used to control the execution order, and $[l_{theBoundCopy} = l_{user}]$ is the precondition of the end of the service, which means the user gets the bound copy. Initially, it is false and it is set to be true along the transition labeled with SD. The first transition labeled with RP and RD is then enabled afterwards.

An abstract service S_i can be realized by a functionally equivalent concrete service. Often, multiple concrete services are available to realize S_i. Given an abstract composite service composed of multiple abstract services $S_1, \ldots, S_n$, we can obtain a concrete composite service by instantiating each abstract service S_i with a concrete one. It is then implied that a concrete composite service can also be expressed in the form of a network of FSM where transition labels are concrete services. For instance, Figure 4 shows the network of FSM representing an instantiation of the abstract composite service shown in Figure 3.

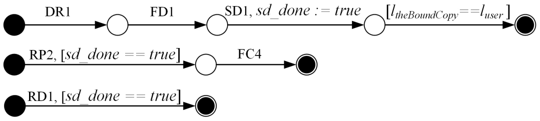

DR1
FD1
SD1, $sd\_done := true$
$[l_{theBoundCopy}==l_{user}]$
RP2, $[sd\_done == true]$
FC4
RD1, $[sd\_done == true]$

Fig. 4. FSM for Concrete Composite Service

Unlike traditional Web service composition where all activities happen in the virtual space, the O2O paradigm connects services taking place in not only the virtual but also the physical space. For instance, invoking an online **Remote Printing** service would result in certain *physical object* (i.e., the printout) being located at certain *physical location*. Furthermore, this online service may be connected to an instance of **Fetching Bound Copy** service taking place offline at the location where the printout is. Intuitively, online services do not require the users to be at specific locations and their executions do not depend on the availability of certain physical objects. For example, the **Document Reservation** service takes as input only the information of a document and it can be invoked anywhere. In contrast, offline services may take place only at certain physical locations and may rely on the existence of certain physical objects at certain physical locations.

In the following, we formalize the interplay between online and offline services. To capture the services' effect on physical objects, without loss of generality, we assume a set of physical objects O and a set of physical locations L. Furthermore, for each object $o \in O$, we define a variable l_o to denote its location. The initial value of l_o is the initial location of o. Formally, we define a concrete service as a 3-tuple $(guard, name, action)$ where $guard$ is the pre-condition on the invocation of the service; $name$ is the name of the service; and $action$ is a program (e.g., assignments) capturing the effect of the service. Here $guard$ may be constituted by the propositions on the value of l_o for certain object o and $action$ may update the value of l_o. For instance, the offline **Fetching Bound Copy** service may have a precondition which requires that the document is at certain station, i.e., $l_{theDoc} = theStation$, whereas the online **Document Reservation** service may set the location of the reserved document to be at certain station. Note that $guard$ and $action$ may have additional propositions. For instance, the **Fetching Bound Copy** service may require a proof of reservation (e.g., a QR code) in its precondition.

Formally, a service $(guard, name, action)$ is online if $guard$ is independent of physical locations (i.e., any variable l_o or location of any agent as introduced later), and is offline otherwise. Both online and offline services however may update some locations l_o, i.e., resulting in certain objects appearing at or moving to certain locations. Given a concrete composite service in the form of a network of FSM,

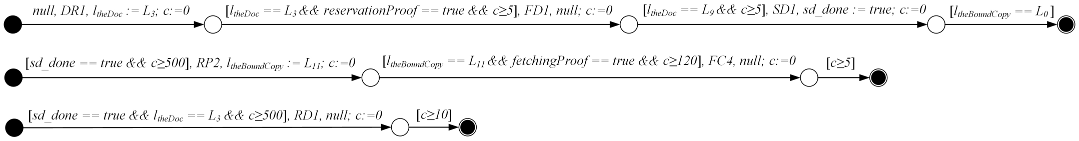

null, DR1, l_{theDoc} := L_3; c:=0
[l_{theDoc} == L_3 && reservationProof == true && c≥5], FD1, null; c:=0
[l_{theDoc} == L_9 && c≥5], SD1, sd_done := true; c:=0
[$l_{theBoundCopy}$ == L_0]
[sd_done == true && c≥500], RP2, $l_{theBoundCopy}$:= L_{11}; c:=0
[$l_{theBoundCopy}$ == L_{11} && fetchingProof == true && c≥120], FC4, null; c:=0
[c≥5]
[sd_done == true && l_{theDoc} == L_3 && c≥500], RD1, null; c:=0
[c≥10]

Fig. 5. Timed Automaton for Concrete Composite Service

replacing each concrete service (i.e., transition label) with the corresponding triple $(guard, name, action)$ turns the network of FSM into a network of timed automata. In order to model concrete services which take certain amount of time to finish, we can introduce an auxiliary clock c to capture the time constraint. For instance, if the concrete service of $name$ takes d time units to finish, we model it as follows. First, we reset c along the transition labeled with the triple $(guard, name, action)$. Assume that the transition leads to a state s. We then constrain all transitions leaving s such that they must satisfy $c \geq d$. Intuitively, this modeling can be interpreted as "when the transition occurs, the concrete service is invoked and it finishes only when the system leaves the state s". We remark that by "timed automata", we do not mean timed automata proposed in [2] but the ones extended with variables as supported in UPPAAL [8]. We denote this network of timed automata representing a concrete composite service as $\mathcal{CS}$. For instance, Figure 5 shows the corresponding $\mathcal{CS}$ for the concrete composite service shown in Figure 4.

Agents An abstract composite service focuses on high-level service requests and is not concerned with how each abstract service is realized. That is, an abstract composite service specifies how services are connected through *time*. In contrast, concrete composite services instantiating an abstract composite service are constrained by their preconditions. In particular, a concrete composite service must deal with how the concrete services are connected through the *physical space*, e.g., in order to invoke the next concrete service, we may need the user to physically relocate certain objects so that its precondition is satisfied. This problem is solved by employing a set of agents A (e.g., people participating in the social collaboration).

In the following, we use A to denote the set of agents. We write l_a where $a \in A$ denotes the location of agent a. Intuitively, each agent is a separate process, who can move from location to location, with or without certain physical objects, in certain amount of time. Formally, an agent is specified in the form of a timed automaton $\mathcal{A}_i = (S_i, init_i, C_i, \Sigma_i, T_i)$ where: S_i is the smallest set containing all locations in L as well as a state $m2n$ for every two ordered locations $\langle m, n\rangle$ denoting the state of transiting from location m to n; $init_i \in L$ is the initial location of the agent; $C_i = \{c_i\}$ contains one and only clock c_i which is used to specify the time cost of the agent moving from one location to another; $\Sigma_i = \{\tau\}$ is an alphabet containing only an invisible τ event (as the name of the events are irrelevant); and T_i is the smallest labeled transition relation satisfying the following conditions.

- For each object $o \in O$ and each state m in $S_i \cap L$, there is a transition $(m, g, \{c_i\}, act, m2n)$ from m to state $m2n$ where g is $l_o = m$ and act is the program $l_o := inTrans$. Note that $inTrans$ is a special constant denoting that the object is not at any location in L but rather in a transition between the locations.
- For each object $o \in O$ and state n, there is a transition $(m2n, g, \{c_i\}, act, n)$ from $m2n$ to n where g is $c_i = d \wedge l_o = inTrans$ and act is the program $l_o = n$. Note that d is a constant denoting the time required for this agent to travel from m to n.

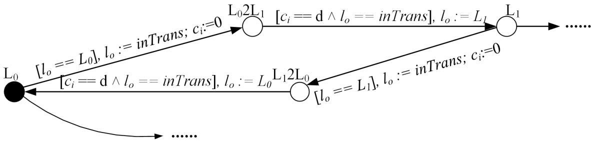

L_0
L_02L_1
L_1
L_12L_0
[l_o == L_0], l_o := $inTrans$; c_i:=0
[c_i == d ∧ l_o == $inTrans$], l_o := L_1
[l_o == L_1], l_o := $inTrans$; c_i:=0
[c_i == d ∧ l_o == $inTrans$], l_o := L_0
......
......

Fig. 6. Timed Automaton for Agent

By definition, when an agent moves from a location to another, it may carry an object along if the object is at the starting location. It is in this way the agents help realizing a concrete composite service, i.e., by connecting concrete services in the physical space. Note that we assume that an agent can each time only bring one object. This assumption can certainly be lifted. For instance, an agent in our motivating example is specified by the timed automaton shown in Figure 6.

A set of agents is defined as the parallel composition of the agents (since they can simultaneously move between locations to help accomplish a concrete service composition more efficiently), written as $\mathcal{A} = \mathcal{A}_1 \parallel \mathcal{A}_2 \parallel \cdots \parallel \mathcal{A}_{|A|}$ where $|A|$ is the number of agents in the system. For the interest of space, we skip the definition of parallel composition of timed automata and refer the readers to [2] for details.

A. Deadline and Cost

An O2O service composition with social collaboration is then defined as the parallel composition of $\mathcal{CS}$ and $\mathcal{A}$, written as $\mathcal{CS} \parallel \mathcal{A}$. Adopting standard definitions in [16], a timed run of $\mathcal{CS} \parallel \mathcal{A}$ is a finite sequence of alternating states and events.

$$\pi = \langle s_0, x_0, s_1, x_1, \cdots, x_{n-1}, s_n\rangle$$

where s_i is a system state which captures the state of the composite service as well as where the agents and objects are; x_i is either a concrete service (i.e., the invocation of the service) or a real number in $\mathbb{R}$ denoting the elapsing of certain x_i number of time units. The total execution time of the run is the accumulated sum of all $x_i \in \mathbb{R}$. The run is rooted if it starts with the initial state of $\mathcal{CS} \parallel \mathcal{A}$. Furthermore, the run is accepting if it is accepting by the composite service FSM network, i.e., it completes the composite service.

A composite service often has a deadline, i.e., the composite service must be completed before certain time. Given a rooted accepting run of $\mathcal{CS} \parallel \mathcal{A}$, we can check whether it satisfies the deadline by checking whether its execution time is within the deadline. We call a rooted accepting run of $\mathcal{CS} \parallel \mathcal{A}$ which satisfies the deadline as an execution of the composite service. Furthermore, we define a cost function $cost$ which, given a concrete service cs, $cost(cs)$ is the cost of invoking that service; and given an agent o and two ordered locations $\langle m, n \rangle$, $cost(o, m, n)$ is the cost of o traveling from m to n. Furthermore, in the context of social collaboration, if each agent is assumed to be traveling from its initial location to a destination, we can measure the additional cost for each agent, in terms of how much extra cost it must bare, i.e., its cost of participating the service execution and then traveling to the destination minus its cost of traveling from the initial location to the destination directly. As a result, given an execution of $\mathcal{CS} \parallel \mathcal{A}$, we can calculate its cost by aggregating the cost of all the steps, as we extract information on what concrete services have been invoked and the additional cost of all the agents participating the execution. Hereafter, we denote the cost of an execution π as $cost(\pi)$.

Problem Definition The problem is then defined as follows. Given an abstract O2O composite service AS composed of abstract services $AS_1, AS_2, \cdots$, and each abstract service can be instantiated with a set of concrete services, identify the most cost-effective execution π of the composite service.

In the following, we briefly analyze the complexity of the problem. Let $|AS_i|$ denote the number of concrete services available to instantiate abstract service AS_i. In general, there are a total of $\Pi_i |S_i|$ possible concrete composite services instantiating an AS. Furthermore, given a concrete composite service, the number of executions are in general exponential in the number of agents. In fact, identifying the most cost-effective execution of a given concrete composite service requires solving a complicated planning problem, which not only has a constraint on the execution time (i.e., the deadline), but also involves concurrency (since all agents are running in parallel). Furthermore, we need to not only find a feasible schedule but rather to find the most cost-effective one. The complexity of our problem is thus very high. In practice, this problem is often solved based on simple ad hoc ways, e.g., not considering social collaboration or assuming that agents work sequentially. In the following, we develop a method which solves this problem in a way such that often a sub-optimal solution can be identified in a reasonable amount of time.

IV. Approach

Given a finite set of concrete services, agents and physical objects, according to our definition, there is always an optimal solution. The complexity in identifying the optimal solution is however extremely high. In this section, we set out to develop a practical approach for solving the problem.

Our approach tackles the complexity at two levels. Given a concrete composite service which is in the form of a network of timed automata, we apply a model checker called UPPAAL-CORA [6] to automatically identify an execution of the composite service with the minimum cost. UPPAAL-CORA is an extension of the well-known UPPAAL model checker [8] for cost optimal reachability analysis, which serves our purpose well. Furthermore, we apply a number of well-founded heuristics to help reduce the state space explored by UPPAAL-CORA significantly. At the level of service selection, we have the combinatorial complexity as there are often multiple concrete services available for each abstract service. We thus apply a general optimization technique, i.e., a version of the MCMC (Markov Chain Monte Carlo [15]) algorithm, so that we can iteratively improve our service selection. We remark that service selection optimization has been well-researched [26], [25], [1], [5]. However, existing approaches do not apply here since our problem additionally requires us to solve the optimal reachability problem given a concrete composite service. In the following, we introduce the two levels of optimization in detail.

A. *MCMC for Service Selection*

The first problem we would like to solve is the problem of combinatorial explosion in the number of concrete composite services. Our remedy is to adopt a version of the MCMC [15], as shown in Algorithm 1, in the hope to find a sub-optimal service selection efficiently.

The inputs of the algorithm include: the abstract composite service AS composed of abstract services $AS_1, AS_2, \cdots$, a set of concrete services CS_i for each AS_i, a finite set of agents A (each of which is in the form of a timed automaton as discussed in Section 3), as well as an optimal threshold on the cost. Algorithm 1 starts with randomly selecting one concrete service $cs_i \in CS_i$ for each abstract service in AS so as to obtain a concrete composite service CS. Next, a timed automata model is built based on the discussion in Section 3, which is then optimized with a set of heuristics for reducing the state space (explained later in this section) and submitted to UPPAAL-CORA in order to identify an "optimal" execution π. At line 4, we compare the cost of the execution π with the given threshold to check whether it is considered acceptable. If it is acceptable, we terminate and report that an acceptable concrete composite service together with an execution which has a cost lower than the threshold has been identified. Otherwise, we proceed to line 5 to search for better concrete composite services. We remark that in the case that the threshold is not provided, we iterate through the loop from line 4 to line 16 until it times out and we report the best execution that has been identified so far.

During the loop, we first check whether we have exhausted all combinations of concrete services. If the answer is yes, we report that no acceptable execution has been identified at line 14. Otherwise, we identify a new concrete composite service, by changing a part of the selection in the best concrete composite service CS identified so far, at line 6. We then identify the "optimal" execution based on the new concrete composite service at line 7. At line 8, we compare the new execution π' with the current best one π. If the new one has a smaller cost, we take π' as the new current best execution at line 9 and similar we take CS' as new best concrete composite service. The condition at line 10 states that even if the new execution is not better than π, there is still certain probability that we might use the new concrete composite service to identify better ones in the future. That is, we generate a random number between 0 and 1, and depending whether the random number is large than a predefined acceptance rate, we decide whether to search better concrete composite service based on CS or CS'. This strategy is related to avoid local minimum and typically the acceptance rate is kept below 0.5.

We remark that the above algorithm is designed based on [3], which has been shown to be effective if the search space is extremely large. Furthermore, it often converges well before the limit is reached [3]. In our setting, this is important as it implies that we do not have to apply UPPAAL-CORA many times in order to identify a good execution of composite service. We acknowledge that the performance of Algorithm 1 is in nature "random" and thus we show empirical study results in Section 5 to show its effectiveness.

Algorithm 1 Service Selection

```
function MCMC($AS$, $\bigcup\{CS_i\}$, $A$, threshold)
    let $CS$ be a concrete composite service obtained through randomly select a concrete service $cs_i \in CS_i$;
    let $\pi$ be an "optimal" execution obtained with UPPAAL-CORA based on $CS$;
    while $cost(\pi) \geq threshold$ do
        if there are still unexplored concrete composite service then
            randomly change some selection in $CS$ to get a new concrete composite service $CS'$;
            apply UPPAAL-CORA on $CS'$ to obtain an "optimal" execution $\pi'$;
            if $cost(\pi') < cost(\pi)$ then
                $\pi := \pi'$; $CS := CS'$;
            else if $random(0,1) < acceptRate$ then
                $CS := CS'$;
            end if
        else
            return $NULL$
        end if
    end while
    return $\pi$
end function
```

B. Optimal Execution Identification

In the following, we explain how an "optimal" execution is identified given a concrete composite service. Given the concrete composite service and the models of the agents, we can systematically build a network of timed automata and apply UPPAAL-CORA to identify the optimal execution. The good news is that with such a model, we are able to systematically explore all possible social collaboration. For instance, it is possible that multiple agents collaboratively deliver an object from one location to another through certain path while each agent is only employed for a segment of the path, i.e., the same object could switch multiple hands before reaching its destination. The bad news is, as a result, the search space is extremely large and thus scalability becomes an issue. Therefore, we propose a number of heuristics for reducing the search space. In the following, we first discuss the heuristics and then present how UPPAAL-CORA is used.

Heuristic 1: Less Changing Hands In the model presented in Section 3, several agents may join in the transportation of one object. This results in a huge number of runs to be explored by UPPAAL-CORA. In practice, if one object at certain location is to be delivered to another location in order to carry out an offline service, it is often the case that only one agent is employed to transport the object so that the object does not change hand half way to its next destination. This is reasonable in practice because changing hands half way would often cause unnecessary complication, e.g., the agent receiving the object next would have to wait at the location for the agent giving away the object. We adopt the same strategy in this work so as to reduce the search space. In particular, we analyze the given concrete composite service to identify the *delivering jobs*. There is a delivering job if there exists a concrete service $(guard, name, action)$ such that $action$ sets an object o to be at certain location, and there is a later concrete service $(guard', name', action')$ such that the guard condition $guard'$ or a precondition of the end of the service requires o to be at a different location and object o is irrelevant to all the concrete services in between. The identification of the *delivering jobs* is done automatically using a path traversing algorithm based on the concrete service model.

For instance, given the composite service presented in Section II, we identify the following delivering jobs. First, a delivering job is needed to fetching the document in a specific library station and transport it to a specific shop for document scanning. Second, a delivering job is needed to fetch the document at where it is scanned and return it to a specific library station. Third, a delivering job is needed to fetch the bound copy at a specific service station of service **Remote Printing**, and transport it to where the user is.

Once the delivering jobs are identified, based on the assumption that each delivering job is to be carried out only by one agent, we know the maximum number of agents required to complete the composite service. Given a set of N agents and K delivering jobs, the question thus becomes how to select K or less agents out of N agents so that one agent is assigned one or more delivering jobs. The total number of such choices is $\binom{K}{N}$. Next, for each such choice, we can use UPPAAL-CORA to identify the optimal execution. We simply take the best execution identified with these choices as the result. We remark that while making $\binom{K}{N}$ calls of UPPAAL-CORA sounds expensive, in general K could be small and furthermore, all these calls of UPPAAL-CORA can be easily parallelized (i.e., using multiple computers to run UPPAAL-CORA with different choices concurrently) and therefore the overall execution time is only determined by the "worse" choice, i.e., the choice which UPPAAL-CORA spends the most time in order to identify the optimal execution. For example, in the composite service presented in Section II, there are 3 delivering jobs which means 3 agents are needed. If there are 5 agents who are willing to collaborate, we choose 3 of them and there are $\binom{3}{5}$ different combinations.

This heuristic helps to reduce the search space significantly since, firstly, we reduce the number of timed automata in the network (i.e., exactly $N - K$ automata less) and, secondly, it allows to revise the agent models to reduce the number of states in each agent model, as we discuss below.

Heuristic 2: Hardworking Agents The model of agents introduced in Section III is intuitive and generic. It however contains many states which translates to high complexity when we use UPPAAL-CORA for identifying the optimal execution. For example, considering the example service composition represented in Section II, there are twelve locations (i.e., L_0 to L_{11}) in total and two objects need to be transported (i.e., **Document** and **Bound Copy**). As a result, the timed automaton model of each agent in the worst case (if the agent can travel to any location) contains 288 states. To explore the social collaboration among several agents, the overall network of timed automata would contain several timed automata with 288 states in addition to the timed automata modeling the service composition, which results in a huge search space.

A closer look reveals the agent model contains many "lazy" behaviors which are unlikely to result in the optimal execution. For instance, an agent could simply wait at certain location for a long time or he/she could move in any direction even if moving towards where the objects are would clearly yield better results. Our second heuristic is thus to refine the agents so as to remove these lazy behaviors, based on the assumption that often having hard-working agents who actively seeks out object delivering tasks would produce better execution. Combined with the first heuristic, this allows us to re-model the agents such that all intermediate states for transporting an object from one location to its next destination can be removed (since the object does not change hand half way).

An example agent model is shown in Figure 7, which is to be understood only with a model of an object in the system, shown in Figure 8. We start with explaining the model modeling an object first. Recall that with the first heuristic, the question is now on how to assign delivering tasks to the agent. It is possible that the same object may be a part of multiple delivering jobs. Theoretically, however, we can simply treat the same object in different delivering jobs as different objects. This is assumption we make in Figure 8, for the sake of readability. In the object model, we have the following parameters: variable cl which represents the current location of the object; src which represents the source location of the object; and des which represents the destination location of the object. Because the same object could be assigned to any of the agents, in the object model, we first have K branches (where K is the number of delivering jobs) which models the choice assigning the object to one of the agents. This is done through a pairwise synchronous message, which is a feature available in UPPAAL-CORA. Intuitively, through a synchronous handshake between the object and one of the agents, the agent is now assigned to deliver this object. Afterwards, we have the state **being generated** and transition such that the object is set to be at its source location. State **being handled** models is the state where the object is in the process of being transported from its source to its destination. Next, we have a synchronous message between the agent and the object again once the object reaches its destination.

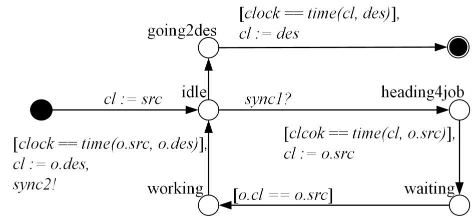


Fig. 7. Revised Timed Automaton for Agent

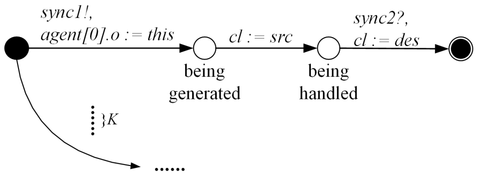


Fig. 8. Timed Automaton for Object

Next, we explain the detail of the revised agent model. In this model, we use the states to represent the status of the agent, e.g., idling when there is no delivering task, heading to the location of the delivering task, etc., and use a number of variables to encode other relevant information. In particular, variable cl represents the current location of the agent; variable src represents the initial location of the agent; and variable des represents the destination location of the agent. Furthermore, function $time(s, d)$ is a function which computes the time required for the agent to travel from location s to location d. Note that given an agent, we can pre-compute function $time(s, d)$ for all pairs of locations.

Initially, the agent's location is set to his initial location (i.e., action $cl := src$). Afterwards, the agent enters **idle** state, where he/she awaits for a delivering job. After the synchronous message with an object, the agent received the delivering job and, without any delay, he/she proceeds to the location of the object. This is modeled as the state **heading4job**. After $time(src, o.src)$ time units where $o.src$ denotes the location of the object, the agent reaches the location of the object. The agent is then at **waiting** state where he might wait for a while until the object is there (i.e., the concrete service which results an object being at the location may take time to execute). After the object is ready at the location (captured by the condition $o.cl$ == $o.src$, i.e. the object's current location is its source location), the agent enters state **working** which means that he is now working on delivering the object. After exactly $time(o.src, o.des)$ time units, the agent sends a synchronous message to the object so as to signal that the object is now delivered to its destination. Afterwards, the agent goes back the **idle** state so that it may be assigned to another delivering job. If the agent has its own destination after all the object delivering, it enters the state **going2des** and takes the respective amount of time to reach its destination and terminates.

This heuristic helps to reduce the number of states in each agent model significantly. In fact, the number of states in the agent model and the object model remain constant. The complexity in identifying the optimal execution is thus largely depending the number of combinations on assigning the objects to different agents. As we have discussed above, this complexity could be easily tamed by using multiple computers to compute the optimal cost of each choice.

V. Evaluation

To evaluate the proposed approach, we conduct an experimental study to answer the following two research questions.

- **RQ1**: Can O2O service composition considering route planning and social collaboration provide better composite services than otherwise, i.e., simply treating offline services as online services or having O2O service composition without social collaboration? (Effectiveness)
- **RQ2**: How scalable is the proposed approach for solving the problem of O2O service composition with social collaboration, in the presence of an increasing number of agents and candidate services? (Scalability)

We have shared the experimental materials online[1], including the Uppaal models, input data, and the implementation of our approach and baseline approaches.

A. Experimental Setup

In order to answer RQ1, we evaluate our approach by solving the composite service **Reproduce Borrowed Document** shown in Section II and compare its performance with two baseline methods. One is an **conventional approach**. In this approach, offline services are treated the same way as online services and thus the optimal service composition is identified through existing service selection approaches [4], [5], [29], [30]. It means that we select each concrete service based on its QoS values without considering route planning or social collaboration. As a result, given a time constraint the same set of concrete services will be always selected for the best concrete composite service. We assume that the user would follow the best route to execute all the selected offline services by himself/herself. The other is an **offline-aware approach**. In this approach, we identify the optimal service execution through concrete service selection as well as considering the route planning for offline services. However, there is no social collaboration and the user himself/herself would execute all the offline services. The difference between the two approaches is on whether the cost of executing offline services is considered during concrete service selection.

In order to answer RQ2, we evaluate the computation time of our proposed approach with different numbers of agents and candidate concrete services. The evaluation is also based on the composite service **Reproduce Borrowed Document** shown in Section II. As the shortest route and traveling time between two locations can be directly obtained, we do not consider the influence of the size of the map on the computation time.

We use the following settings for both RQ1 and RQ2. The time each agent needs to travel between two locations is shown in Table I. For example, traveling from L_0 to L_1 or L_1 to L_0 needs 200 time units. The cost of agent is set as 0.01 per time unit. For example, the cost of one agent transporting one object from L_0 to L_1 is $200 \times 0.01 = 2$.

TABLE I
Time Required to Travel between Locations

Loc.	L_0	L_1	L_2	L_3	L_4	L_5	L_6	L_7	L_8	L_9	L_{10}	L_{11}
L_0	0	200	400	400	220	480	660	660	940	820	1120	960
L_1	-	0	200	200	360	420	460	600	760	760	920	900
L_2	-	-	0	400	500	240	260	420	560	580	720	720
L_3	-	-	-	0	480	220	660	400	680	560	840	700
L_4	-	-	-	-	0	260	760	440	720	600	880	740
L_5	-	-	-	-	-	0	500	180	460	340	620	480
L_6	-	-	-	-	-	-	0	580	300	760	460	620
L_7	-	-	-	-	-	-	-	0	280	520	440	600
L_8	-	-	-	-	-	-	-	-	0	460	160	320
L_9	-	-	-	-	-	-	-	-	-	0	260	140
L_{10}	-	-	-	-	-	-	-	-	-	-	0	120
L_{11}	-	-	-	-	-	-	-	-	-	-	-	0

The cost threshold in Algorithm 1 is set as the optimal cost identified using the **offline-aware approach** with the same initial user location. For example, if the user is at L_x and the optimal cost identified using the **offline-aware approach** is y, y is set as the cost threshold for our approach for the cases where the user is initially at L_x. This way, we can evaluate whether our approach can identify better executions in a reasonable amount of time. We remark that setting the threshold to be a constant value independent of the initial user location is not ideal because the cost could vary significantly with different initial user location. The accept rate used at line 10 of Algorithm 1 is fixed as 0.1 heuristically.

All the experiments are conducted on the same machine with a Dual-core 2.9GHz CPU and 8 GB RAM. The version of UPPAAL-CORA used in the experiments is 060910.

B. Effectiveness (RQ1)

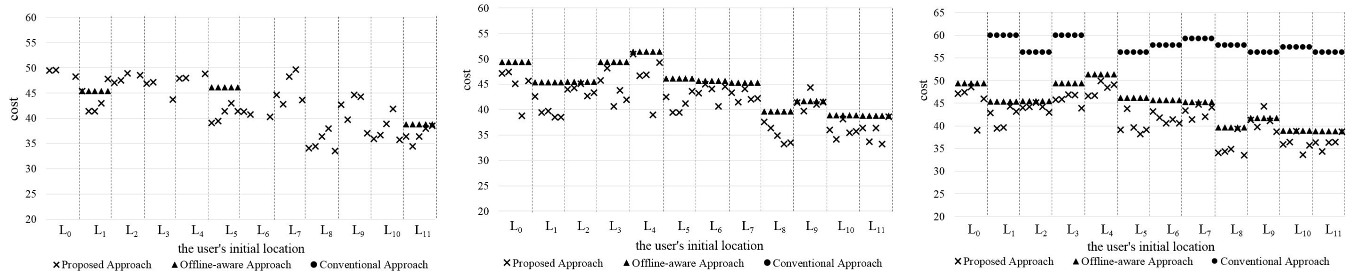
We evaluate the performance of the three approaches under different time constraints. For each time constraint, we conduct twelve experiments for each of the three approaches. In each experiment, the user is at a different initial location in the map of Figure 2 (i.e., L_0 to L_{11}). We apply the three approaches to identify an optimal solution, i.e., a concrete composite service and an execution plan of it which has the minimum cost. Note that the cost includes the cost for both concrete services and the cost of agents traveling between different locations.

In order to evaluate our proposed approach in the presence of multiple agents, we generate five different groups of agents. Each group contains five agents (including the user himself/herself) with random initial (and destination) locations. We apply these five agent groups in all the twelve experiments, and compute the cost of the composite service with each agent group. Note that in each experiment the two baseline approaches produce the same results for all the agent groups as they do not consider social collaboration. The time out of Algorithm 1 is set to be 120 seconds.

Figure 9 shows the optimal cost identified by the above-mentioned three approaches under different time constraints (1200, 2400, 3600 time units). The X axis denotes the different initial locations of the user and the five groups of agents, and the Y axis denotes the identified optimal cost. If a point is missing, it means that the corresponding approach can not find a solution with the given initial location and agent group.

From the results, it can be seen that, when the time constraint is tight (1200 time units) our approach can find a solution in 85% cases, while the **offline-aware approach** can only find a solution in 25% cases and the **conventional approach** can find a solution in none of the cases; when the

[1] https://o2oservice.wixsite.com/o2o-service-sample



(a) time constraint = 1200 time units (b) time constraint = 2400 time units (c) time constraint = 3600 time units

Fig. 9. Performance of the Three Approaches under Different Time Constraints

time constraint is less tight (2400 time units) both our approach and the **offline-aware approach** can find a solution in all the cases, while the **conventional approach** fails to find a solution in any of the cases; when the time constraint is loose (3600 time units) both our approach and the **offline-aware approach** can find a solution in all the cases, and the **conventional approach** can find a solution in 83.3% cases. On average, our approach improves the solution of the **conventional approach** by 29.8% with a median of 29.2% and improves the **offline-aware approach** by 7.2% with a median of 8.6% when both approaches can find a solution.

Although the solution found by our approach is almost always the best among all the three approaches, we can also observe that in several cases the solution found by the **offline-aware approach** is even better than the solution found by our approach, including a case of L_1 in Figure 9(a), a case of L_9 in Figure 9(b), and a case of L_9 in Figure 9(c). There are two reasons for this. First, the MCMC algorithm we use in our approach works by random walking through the search space and thus it does not guarantee that every time the solution is better with collaborative agents. Second, our approach needs more time than the **offline-aware approach**. As a result, a solution with a cost more than the threshold may be returned when timeout occurs despite that there could be other better solutions with less cost.

The solutions found by our approach involve one to three agents. The average numbers of agents involved in the solutions are 2.33, 1.83, 1.78 when the time constraints are 1200, 2400, 3600 time units. *The result suggests that when the time constraint is tight, social collaboration becomes important and our approach is able to find such collaborative solutions.*

The above results are expected as the **conventional approach** completely ignores the route planning, which is essential for O2O service composition, and thus always has the highest cost or fails to find a solution. Our approach takes into consideration the route planning for offline services, and takes advantage of social collaboration to further reduce the cost, and thus nearly always has the best cost.

The above analysis answers RQ1 positively that our approach nearly always results in better O2O service composition solution than the other two alternative approaches.

C. Scalability (RQ2)

For a given number of agents and a given number of concrete services of each abstract service, we run our approach with 60 different settings by considering different initial user locations (from L_0 to $L_1$1), each with five different groups of agents with random initial (and destination) locations. We take the average computation time of the 60 settings as the result. In all the experiments, the time constraint is set as 3600 time units. The time out of Algorithm 1 is set to be 600 seconds.

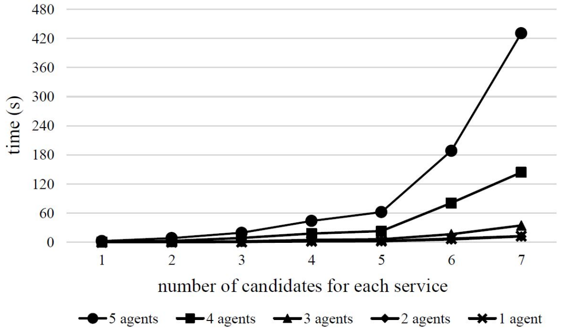


Fig. 10. Computation Time of the Proposed Approach

Figure 10 shows the change of the time used to identify the optimal solution using our approach. The X axis denotes the number of candidate concrete services of each abstract service and the Y axis denotes the time (by seconds). Each curve in Figure 10 represents a different number of agents (from one to five). It can be observed that with the increase of the number of agents and candidate services our approach takes more time to find an optimal solution. With one to three agents, it takes 0.1s to 34.7s to generate a solution when the number of candidates for each service increases from one to seven. With four agents, it takes 0.9s to 144.1s. With five agents, it takes 2.5s to 430.8s. While indeed state space explosion would eventually happen, thanks to our heuristics presented in the previous section, the time increases steadily rather than exponentially. In practice, we believe that having a social collaboration with five agents is reasonably sufficient for most O2O service composition. Because often the more agents, the more likely the social collaboration becomes chaotic and unreliable. For example, certain agents, i.e., human beings, may fail to execute an offline service as planned when some accidents happen, and more agents involved in the collaboration are more likely resulting in failing the service process. In our experiments, the proposed approach could give a solution with five agents in no more than one minute on average when each abstract service has up to five candidates. This is acceptable considering a composite service involving offline services usually takes dozens of minutes to several hours. Moreover, if we parallelize the MCMC algorithm, the time for generating one solution can be further shortened.

The above analysis answers RQ2 positively that our approach performs well with an increasing number of agents. The time for generating one solution grows if the number of agents increases. Nonetheless, considering that five agents may be sufficient for most O2O service composition scenarios, the time for generating one solution is sufficiently fast.

VI. Discussion

The composition of online to offline services in practice can be complicated by the uncertain nature of offline services. Currently we assume that the availability and execution time of an offline service are known in composition planning, but they may be uncertain and continuously changing. For example, the time required to get an offline service may depend on how many customers are waiting for it. So is the traveling time of agents. Currently, we assume that the time required for an agent to travel between two locations can be predicted before hand. In practice, it heavily depends on the traffic on the way, and the bus/train schedule if public transportation is chosen.

On the other hand, the social collaboration involved in the composition of O2O services can be complicated by social issues in practice. People participating in the collaboration need to share their routes with the platform, but not necessarily with others. They can remain anonymous when participating others' tasks except that they need to deliver objects with others face to face. To encourage people to participate in social collaboration, it is required that some kind of incentive mechanism is established. For example, people can get bonus points for their efforts, which can be used in the future to recruit others for help.

Based on the above analysis, we foresee that large-scale application of O2O service composition can be achieved by integrating it with, and supplying it on, a location-based live service platform with mobile social networking. With the platform, offline services can be registered with their locations on the map and users can use its map service to navigate to destinations. Based on the locations of offline services and the destinations of users, the platform can plan collaboration solutions and invite users to participate. On the other hand, a user can use the social networking service to limit the scope of collaboration for privacy.

Furthermore, the platform can connect and analyze big data from a wide range of sources (e.g., vehicles, traffic surveillance cameras) to construct customer-oriented composite services as suggested in [28]. Among the sources there can be crowd sensing data from the users. For example, they can report whether an offline service is available and how many customers are waiting for it. With this kind of big data, the platform can not only predict the execution time of offline services and traveling time on the way but also dynamically adapt the composition and execution plan according to real-time information.

VII. Related Work

Traditional service composition research focuses on the combinational selection of component services by local selection [7], [19], global optimization [4], [5], [29], [30], or a combination [1]. Some research [24] synthesizes local QoS constraints (e.g., response time) of component services that guarantee the global QoS requirements. O2O service composition is more than selecting an optimal set of component services. For each concrete composite service, O2O service composition needs to further identify a cost-effective execution by finding an optimal collaboration among involved agents.

Researchers have seen the necessity of integrating virtual (such as cloud services) and physical (such as public transportation) services from various domains to produce composite service solutions to meet customer requirements [28]. The current research is mainly focused on physical services in a local environment [13], [23]. This kind of device and sensor services can be regarded as offline services, since they can only be accessed in certain locations, but the service composition considers local services instead of geographically distributed offline services.

O2O service composition is also different from the so-called location based services (LBS), which provide personalized and context-aware services according to the geographic location of a user or other entity [18], [10]. Location based services are usually online services that can be accessed anywhere but may provide different information (e.g., advertisement of a nearby restaurant) depending on users' locations.

Related to this work in self-adaptive software systems, Bennaceur et al. [9] propose a vision to support collaborative security by mediating emerging components in an environment. This work adds to the vision that social collaboration could happen among people when their constraints can be formally modeled and reasoned about.

Recently, there has been increasing interest in leveraging large-scale collaboration for problem solving by crowdsourcing [12], [20]. In the area of crowd sensing [21], [27], [31], mobile users in a large scale are involved to contribute sensing data (e.g., position, air pollution) using their mobile phones. The collaboration in this kind of crowdsourcing is targeted at data collection and information acquisition and does not involve location-based execution planning.

VIII. Conclusion

O2O commerce is driving the need of constructing customer-oriented composite services by seamlessly composing online and offline services. In this paper, we have proposed an approach to O2O service selection and composition which incorporates offline route planning and collaboration among the social agents. The approach has been evaluated using a simulated scenario and the results have confirmed its effectiveness and scalability. As future work we foresee the integration of O2O service composition with a location-based real life service platform to enable customer-oriented composite service planning and adaptation.

IX. Acknowledgments

This work is supported by National High Technology Development 863 Program of China under Grant No. 2015AA01A203.

REFERENCES

[1] Mohammad Alrifai and Thomas Risse. Combining global optimization with local selection for efficient QoS-aware service composition. In *Proceedings of the 18th International Conference on World Wide Web*, WWW 2009, pages 881–890, New York, NY, USA, 2009.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1-2):5–43, 2003.

[4] Danilo Ardagna and Barbara Pernici. Global and local QoS constraints guarantee in web service selection. In *Proceedings of the 12nd IEEE International Conference on Web Services*, ICWS 2005, pages 805–806, Washington, DC, USA, 2005.

[5] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *IEEE Transactions on Software Engineering*, 33(6):369–384, 2007.

[6] Gerd Behrmann and Ansgar Fehnker. Efficient guiding towards cost-optimality in UPPAAL. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS 2001, pages 174–188, Genova, Italy, 2001.

[7] Boualem Benatallah, Quan Z. Sheng, Anne H. H. Ngu, and Marlon Dumas. Declarative composition and peer-to-peer provisioning of dynamic web services. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE 2002, pages 297–308, San Jose, CA, USA, 2002.

[8] Johan Bengtsson, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL - a tool suite for automatic verification of real-time systems. In *Proceedings of the DIMACS/SYCON Workshop, Hybrid Systems III: Verification and Control*, pages 232–243, New Brunswick, NJ, USA, 1995.

[9] Amel Bennaceur, Arosha K. Bandara, Michael Jackson, Wei Liu, Lionel Montrieux, Thein Than Tun, Yijun Yu, and Bashar Nuseibeh. Requirements-driven mediation for collaborative security. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2014, pages 37–42, Hyderabad, India, 2014.

[10] Subhankar Dhar and Upkar Varshney. Challenges and business models for mobile location-based services and advertising. *Communications of the ACM*, 54(5):121–128, 2011.

[11] Wayne Duggan. What Does O2O Mean for the Future of E-Commerce? http://tinyurl.com/hoob7m4.

[12] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. CrowdDB: answering queries with crowdsourcing. In *Proceedings of the 38th ACM SIGMOD International Conference on Management of Data*, SIGMOD 2011, pages 61–72, Athens, Greece, 2011.

[13] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Transactions on Services Computing*, 3(3):223–235, 2010.

[14] Juho Hamari, Mimmi Sjöklint, and Antti Ukkonen. The sharing economy: Why people participate in collaborative consumption. *Journal of the Association for Information Science and Technology*, 2015, in press.

[15] W.K Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[16] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Timed transition systems. In *Proceedings of REX Workshop, Real-Time: Theory in Practice*, pages 226–251, Mook, The Netherlands, 1991.

[17] The Wall Street Journal. Amazon's Next Delivery Drone: You. http://tinyurl.com/h29tb36.

[18] Iris A. Junglas and Richard T. Watson. Location-based services. *Communications of the ACM*, 51(3):65–69, 2008.

[19] Fei Li, Fangchun Yang, Kai Shuang, and Sen Su. Q-peer: a decentralized QoS registry architecture for web services. In *Proceedings of the 5th International Conference on Service-Oriented Computing*, ICSOC 2007, pages 145–156, Vienna, Austria, 2007.

[20] Hongwei Li, Bo Zhao, and Ariel Fuxman. The wisdom of minority: discovering and targeting the right group of workers for crowdsourcing. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW 2014, pages 165–176, Seoul, Republic of Korea, 2014.

[21] Moo-Ryong Ra, Bin Liu, Tom F. La Porta, and Ramesh Govindan. Medusa: a programming framework for crowd-sensing applications. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys 2012, pages 337–350, Ambleside, United Kingdom, 2012.

[22] Alex Rampell. Why Online2Offline Commerce Is A Trillion Dollar Opportunity. http://tinyurl.com/28fmpgz.

[23] Thanos G. Stavropoulos, Dimitris Vrakas, and Ioannis P. Vlahavas. A survey of service composition in ambient intelligence environments. *Artificial Intelligence Review*, 40(3):247–270, 2013.

[24] Tian Huat Tan, Étienne André, Jun Sun, Yang Liu, Jin Song Dong, and Manman Chen. Dynamic synthesis of local time requirement for service composition. In *Proceedings of the 35th International Conference on Software Engineering*, ICSE 2013, pages 542–551, San Francisco, CA, USA, 2013.

[25] Tian Huat Tan, Manman Chen, Étienne André, Jun Sun, Yang Liu, and Jin Song Dong. Automated runtime recovery for QoS-based service composition. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW 2014, pages 563–574, Seoul, Republic of Korea, 2014.

[26] Tian Huat Tan, Yinxing Xue, Manman Chen, Jun Sun, Yang Liu, and Jin Song Dong. Optimizing selection of competing features via feedback-directed evolutionary algorithms. In *Proceedings of the 15th International Symposium on Software Testing and Analysis*, ISSTA 2015, pages 246–256, Baltimore, MD, USA, 2015.

[27] Agustinus Borgy Waluyo, David Taniar, Bala Srinivasan, and Wenny Rahayu. Mobile query services in a participatory embedded sensing environment. *ACM Transactions on Embedded Computing Systems*, 12(2):31:1–31:24, 2013.

[28] Xiaofei Xu, Quan Z. Sheng, Liang-Jie Zhang, Yushun Fan, and Schahram Dustdar. From big data to big service. *IEEE Computer*, 48(7):80–83, 2015.

[29] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *Proceedings of the 12nd International Conference on World Wide Web*, WWW 2003, pages 411–421, New York, NY, USA, 2003.

[30] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, 2004.

[31] Pengfei Zhou, Yuanqing Zheng, and Mo Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, MobiSys 2012, pages 379–392, Ambleside, United Kingdom, 2012.